NPS ARCHIVE
2000.09
GHYZEL, P.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

**VISION-BASED NAVIGATION
FOR
AUTONOMOUS LANDING OF
UNMANNED AERIAL VEHICLES**

by

Paul A. Ghyzel

September 2000

Thesis Advisors: Isaac. I. Kaminer
Oleg A. Yakimenko

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2000	Engineer's Thesis

4. TITLE AND SUBTITLE
Vision-Based Navigation for Autonomous Landing of Unmanned Aerial Vehicles

5. FUNDING NUMBERS

6. AUTHOR(S)
Ghyzel, Paul A.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The role of Unmanned Aerial Vehicles (UAV) for modern military operations is expected to expand in the 21st Century, including increased deployment of UAVs from Navy ships at sea. Autonomous operation of UAVs from ships at sea requires the UAV to land on a moving ship using only passive sensors installed in the UAV. This thesis investigates the feasibility of using passive vision sensors installed in the UAV to estimate the UAV position relative to the moving platform. A navigation algorithm based on photogrammetry and perspective estimation is presented for numerically determining the relative position and orientation of an aircraft with respect to a ship that possesses three visibly significant points with known separation distances. Original image processing algorithms that reliably locate visually significant features in monochrome images are developed. Monochrome video imagery collected during flight test with an infrared video camera mounted in the nose of a UAV during actual landing approaches is presented. The navigation and image processing algorithms are combined to reduce the flight test images into vehicle position estimates. These position estimates are compared to truth data to demonstrate the feasibility of passive, vision-based sensors for aircraft navigation. Conclusions are drawn, and recommendations for further study are presented.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Unmanned Aerial Vehicle, Navigation, Infrared Imaging, Image Processing, MATLAB®, Simulation	132
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

VISION-BASED NAVIGATION FOR AUTONOMOUS LANDING OF UNMANNED AERIAL VEHICLES

Paul A. Ghyzel
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1989

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING
and
AERONAUTICAL AND ASTRONAUTICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL
September 2000

NPS ARCHIVE
2000.09
GHYZEL,P.

TK453
G34/415
C.1

THIS PAGE INTENTIONALLY LEFT BLANK

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

ABSTRACT

The role of Unmanned Aerial Vehicles (UAV) for modern military operations is expected to expand in the 21st Century, including increased deployment of UAVs from Navy ships at sea. Autonomous operation of UAVs from ships at sea requires the UAV to land on a moving ship using only passive sensors installed in the UAV. This thesis investigates the feasibility of using passive vision sensors installed in the UAV to estimate the UAV position relative to the moving platform. A navigation algorithm based on photogrammetry and perspective estimation is presented for numerically determining the relative position and orientation of an aircraft with respect to a ship that possesses three visibly significant points with known separation distances. Original image processing algorithms that reliably locate visually significant features in monochrome images are developed. Monochrome video imagery collected during flight test with an infrared video camera mounted in the nose of a UAV during actual landing approaches is presented. The navigation and image processing algorithms are combined to reduce the flight test images into vehicle position estimates. These position estimates are compared to truth data to demonstrate the feasibility of passive, vision-based sensors for aircraft navigation. Conclusions are drawn, and recommendations for further study are presented.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1

A. VISION-BASED NAVIGATION 1
B. PURPOSE 3

II. NAVIGATION USING PASSIVE SENSORS 5

A. POSITION AND VELOCITY ESTIMATION 5
1. Inertial Navigation System (INS) 5
2. Global Positioning System (GPS) 6
3. Air Data Systems 6
4. Vision-based Sensors 7
B. COORDINATE SYSTEMS 7
1. Geodetic Coordinate System 7
2. Earth Centered Earth Fixed 8
3. Local Tangent Plane 8
4. Body Reference Frame 9
5. Camera Reference Frame 11
6. Image Plane Reference Frame 12

III. THREE POINT POSE ESTIMATION PROBLEM 13

A. PROBLEM FORMULATION 13
B. GRUNERT'S SOLUTION 14
C. NUMERICAL THREE-POINT ALGORITHM 16
1. Problem Formulation 16
2. Numerical Algorithm 18

IV. IMAGE PROCESSING 21

A. PURPOSE 21
B. METHODOLGY 21
1. Bisection Thresholding 23
2. Polynomial-Difference 29
3. Sliding Average 38
C. COMPARISON OF METHODS 40
1. Initial Coordinate Determination 40
2. Successive Coordinate Determination 42
D. IMPLEMENTATION 49
1. Introduction 49
2. Image Capture and Storage 51
3. Program Structure 51
4. Image Loading and Normalization 52
5. First Image 53
6. Successive Images 58

7. Image Plane Coordinates 61
8. Data Transformation and Storage 65
9. Integration with the Three Point Algorithm 66

V. FLIGHT TEST 69

A. UNMANNED AERIAL VEHICLE DESCRIPTION 69
1. Airframe Description 69
2. Sensor Description 70
B. FLIGHT PROFILES 74

VI. RESULTS 77

A. POST FLIGHT PROCESSING 77
B. SPOT LOCATION 77
C. COMPARISON WITH DGPS 78

VII. CONCLUSIONS AND RECOMMENDATIONS 81

A. CONCLUSIONS 81
1. General 81
2. Specific 81
B. RECOMMENDATIONS 83
1. Future Investigations 83
2. Flight Test 83
3. UAV Enhancements 84

VIII. REAL-TIME CONTROLLER HARDWARE INTERFACE 85

A. INTRODUCTION 85
B. SUMMARY OF PREVIOUS WORK 86
C. REAL-TIME CONTROLLER 87
1. Rapid Flight Test Prototype System Ground Station 87
2. MATRIX-X® Software Family 88
3. AC-104 System Description 90

LIST OF REFERENCES 95

APPENDIX A. DOCUMENTED MATLAB CODE 97

INITIAL DISTRIBUTION LIST 115

LIST OF FIGURES

Figure 1. ECEF and Local Tangent Plane Coordinate System [Ref. 3] 9
Figure 2. Aircraft Body Reference Frame [Ref. 4] 10
Figure 3. Camera Reference Frame [Ref. 3] 11
Figure 4. Image Plane Reference Frame [Ref. 3] 12
Figure 5. Geometry of the Three Point Pose Estimation Problem [Ref. 1] 13
Figure 6. Intersection of 3 Elliptic Cylinders to Yield 8 Solutions [Ref. 1] 17
Figure 7. Unfiltered 'picture25.bmp' 23
Figure 8. Pixel Luminance Contour of Three Spots of Interest and Surroundings 24
Figure 9. "Hotspot" Pixel Groupings and Center Locations 29
Figure 10. Representative Polynomial Fit to Pixel Luminance Data (Unfiltered) 30
Figure 11. Composite Polynomial Fit-Difference Plot 31
Figure 12. Weibull Row Filter 33
Figure 13. Normal Gaussian Column Filter 34
Figure 14. Composite Image Filter 35
Figure 15. Filtered 'picture25.bmp' 36
Figure 16. Representative Polynomial Fit to Pixel Luminance Data (Filtered) 37
Figure 17. Pixel Intensities vs. Column Position, Polynomial Difference Method 38
Figure 18. Sliding Average Method Performance 44
Figure 19. Polynomial Difference Method Performance 45
Figure 20. Bisection Thresholding Method Performance 47
Figure 21. Time Required to Process Each Image in Landing Approach Video Clip 48
Figure 22. Representative Processed Image with Brightest Spots Indicated 55
Figure 23. Representative Image with Five Brightest Spots Indicated 56
Figure 24. Representative Image with the Three Spots of Interest Indicated 57
Figure 25. Search Box Expansion for Representative Video Clip 60
Figure 26. Search Box Size for Representative Video Clip 61
Figure 27. Final Threshold at Which Three Points Isolated 64
Figure 28. US Army FOG-R Unmanned Air Vehicle, Side View 69
Figure 29. Pixel Height/Width vs. Range from Target 71
Figure 30. IR Camera and VTR Installation in FOG-R UAV 73
Figure 31. DGPS Installation in FOG-R UAV 74
Figure 32. Spot Position and Processing Box Borders For Representative Approach 78
Figure 33. Comparison of IR and DGPS Position in Local Tangent Plane (2-D) 79
Figure 34. Comparison of IR and DGPS Position in Local Tangent Plane (3-D) 80
Figure 35. RealSim Graphical User Interface [Ref. 18] 89
Figure 36. AC-104 Front Panel Arrangement 90

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENT

I would like to thank Professors Isaac Kaminer and Oleg Yakimenko for their patient guidance and thoughtful insight throughout this effort. I would also like to thank Professor Edward Wu for his interest in and cogent analysis of the image filtering portion of this project. Additionally, I am also grateful for the technical advice and consultation provided by Mr. Jerry Lentz and Mr. Don Meeks, without whom the flight test would not have been possible.

Finally, I would like to thank my wife, Therese, and my children, John and Elizabeth, for their unlimited patience, understanding, and encouragement.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. VISION-BASED NAVIGATION

The role of Unmanned Aerial Vehicles (UAV) for modern military operations is expected to expand in the 21st Century, including increased deployment of UAVs from Navy ships at sea. Currently, UAVs are guided by remote control, or they fly a pre-programmed route. Neither case is entirely compatible with operations at sea. Dependence on remote control limits the range at which the UAV can be operated from the controlling platform. A pre-programmed route of flight can be utilized for UAV operations from a fixed takeoff and recovery site but is not compatible with landing on a moving ship at sea if the ships position at landing is undetermined when the UAV is programmed with the route of flight. To facilitate ship-based UAV launch and recovery operations without restricting the ship's freedom to maneuver during the UAV mission, the UAV must be able to determine within acceptable error limits its position, velocity, and attitude relative to the ship on which it will land. Data link systems that provide such relative position information have been employed on Navy ships for operations with manned aircraft. However, these systems require additional transmit and receive equipment to be installed on the ship and the aircraft. The cost, weight, volume, and susceptibility to electronic warfare of these shipboard systems makes it desirable for the aircraft to rely only on passive sensors installed on the aircraft for landing. While DGPS and inertial measuring sensors can provide accurate vehicle position, velocity, and attitude estimates, they are incapable of determining position, velocity, and attitude estimates relative to a moving landing platform. A vision sensor, mounted on the aircraft

in such a manner that the moving ship remains within the sensor field of view while the aircraft is landing can provide such information. Being able to land autonomously on a moving ship passively without reliance on specialized systems installed on the ship removes a significant limitation to the flexibility of maritime UAV operations.

Detection of a ship on the sea surface using vision sensors requires that the ship display visible features that contrast with its surroundings. Visible light cameras are capable of locating a ship on the sea surface at great ranges in ideal, daylight conditions, but they are severely limited at night and in poor weather. Infrared (IR) radiation in the 8-12 micron range can be detected by IR cameras regardless of ambient light, and all ships generally radiate strongly in these wavelengths due to their engine and smokestack. These "hot" features typically contrast well with the "cool" water surrounding the ship due to this significant temperature difference. At the sensitivity limit of a given IR sensor, a ship's IR signature will most likely appear as a single point. As the range between the sensor and the ship is reduced, multiple hot points can typically be resolved. If the actual distances between these points on the ship are known, then the relative location of these hot points with respect to each other in the image produced by the IR sensor can be used to determine the camera's range and orientation relative to the ship. However, vision-based sensors are sensitive to occlusions and times when the points of interest are not in the sensor field of view, referred to here as out-of-frame events. When combined with the navigation data provided by DGPS and inertial sensors, these vision-based position estimates complete the navigation solution required for a UAV to perform an autonomous shipboard landing using only passive sensors. [Ref. 1]

B. PURPOSE

The primary purpose of this thesis is to demonstrate how an aircraft's position, relative to a moving ship can be estimated to support navigation, guidance, and control for autonomous shipboard landing using only data available from a video sensor mounted in the aircraft's nose. The development and implementation of original image processing algorithms that extract relevant reference data from video imagery are presented. A new navigation algorithm that estimates aircraft position, velocity, and attitude based on three known visual reference points is discussed. Integration of the original image processing algorithms with the navigation algorithm is presented. The integrated algorithms are used to produce position estimates from video imagery obtained during actual flight test, and comparison of the vision-based estimates with a Differential Global Positioning System (DGPS) navigation solution is presented.

THIS PAGE INTENTIONALLY LEFT BLANK

II. NAVIGATION USING PASSIVE SENSORS

A. POSITION AND VELOCITY ESTIMATION

The task of automatically landing an aircraft on a pitching ship at sea requires exceptional positioning accuracy in three dimensions that is updated rapidly [Ref. 2]. The available passive sensor suite for modern aircraft includes the Inertial Navigation System (INS), the Global Positioning System (GPS), air data systems, and vision-based sensors [Ref. 1]. Each of these systems individually provides either a high update rate or accurate position and velocity estimates, but no single one provides the highly precise, rapidly updated position estimates that are required to support the autonomous landing of an aircraft on a moving ship at sea [Ref. 2]. By using the techniques of optimal estimation, the strengths of each individual sensor can be blended to produce optimal estimates of position and velocity that are valid over a large frequency range [Ref. 1].

1. Inertial Navigation System (INS)

Probably the most popular long-range navigation system in use is the INS. In the strapdown configuration, this system measures thrust accelerations and angular velocities in the body reference frame and integrates them to estimate velocity and position, and it also determines aircraft attitude. It is entirely self-contained on the aircraft that carries it. While INS position estimates can be updated several times each second, INS systems are subject to numerous bias and drift errors that cause their accuracy to degrade over time. As a result, its accuracy is insufficient for use as a stand-alone navigation sensor for autonomous shipboard landing. [Ref. 2]

2. Global Positioning System (GPS)

The Global Positioning System is a satellite-based radio navigation system with the capability to provide locating data to an unlimited number of users. Most GPS receivers currently available on the market compute precise vehicle position, velocity, and altitude estimates, but GPS is not immune from errors either. Errors that affect the GPS include: atmospheric delays, Selective Availability, clock differences, ephemeris error, multipath, receiver noise, and Dilution of Precision [Ref. 2]. GPS can still be made more accurate by augmenting it with a differential correction provided by either an additional GPS receiver whose exact position is known or via a commercially available satellite service. Even so, its update rate is significantly less than that of the INS, so it too is insufficient as a stand-alone navigation sensor for autonomous shipboard landing. [Ref. 2]

3. Air Data Systems

Air data systems typically consist of a pitot-static system that senses ambient static pressure, total pressure, and outside air temperature. These values are used by an air data computer to estimate aircraft airspeed, altitude, and vertical velocity. Depending on how the pitot-static plumbing is installed, these systems can have a tendency to lag, and there is no way to extract aircraft position from an air data system alone. Clearly, air data systems are also insufficient as a stand-alone navigation sensor for autonomous shipboard landing.

4. Vision-based Sensors

Vision sensors rely on data gleaned from imagery to perform navigation functions. Their greatest contribution to solving the autonomous shipboard landing problem is that they can be used to estimate the aircraft's position relative to the ship based on how the ship appears in the imagery. This is a contribution that none of the previously mentioned sensor systems is capable of, yet it is key to landing on a moving platform. Unfortunately, vision sensors are sensitive to occlusions and occasions when one of the visibly significant reference points is not present in the image frame, hereafter referred to as an out-of-frame event. [Ref. 1]

B. COORDINATE SYSTEMS

The use of vision sensors for aircraft navigation requires the use of several different coordinate systems, including Earth Centered Earth Fixed (ECEF), geodetic, Local Tangent Plane (LTP), Body Reference, Image Plane (IM) , and Camera Reference, and transformations between them. [Ref. 2]

The geodetic and LTP systems depend on the model of the earth's surface. The current standard is the WGS-84 ellipsoid, which is generated by rotating an ellipse with a semi-major axis of 6378137.0 meters and semi-minor axis of 6356752.3 meters about its minor axis. The true north and south poles are the endpoints of the ellipsoid's minor axis. [Ref. 2]

1. Geodetic Coordinate System

The output of navigation systems used on modern aircraft is generally resolved in the geodetic coordinate system, i.e., the output is in terms of latitude, longitude, and

altitude. In the geodetic coordinate system the elevation angle or latitude, λ, is the angle between the ellipsoidal normal and its projection in the equatorial plane. The longitude, ϕ, is the angle in the equatorial plane from the prime meridian (0° longitude) to the given point. The altitude or geodetic height, h, is the distance along the ellipsoid normal from the surface of the earth to the given point.

2. Earth Centered Earth Fixed

The Earth Centered Earth Fixed (ECEF) system is a right-hand Cartesian system with its origin at the center of the earth. The positive x-axis passes from the origin through the intersection of the equator (0° latitude) and the prime meridian (0° longitude). The positive Y-axis passes from the origin through 90° E longitude, and the positive Z-axis passes from the origin through the true north pole. The ECEF system rotates with the earth, but it is independent of the mathematical model of the earth's surface. The ECEF system is presented in figure 1. [Ref. 2]

3. Local Tangent Plane

The local tangent plane (LTP) or local geodetic system is defined by a plane that is tangent to the earth's surface; the point of tangency is the origin of the system. The positive x-axis is in the plane and points to true north; the positive y-axis is in the plane and points to true east. The z-axis is perpendicular to the plane and passes through the origin. For the purposes of this investigation, the positive z-axis points toward the center of the earth, resulting in a right-hand Cartesian system that is also referred to as North-East-Down (NED) and is consistent with the body reference frame. Typically, pure

inertial systems navigate in a local tangent plane coordinate system before outputting position in geodetic coordinates. This frame is also referred to as the universal frame. The local tangent plane coordinate system is presented in figure 1. [Ref. 3]

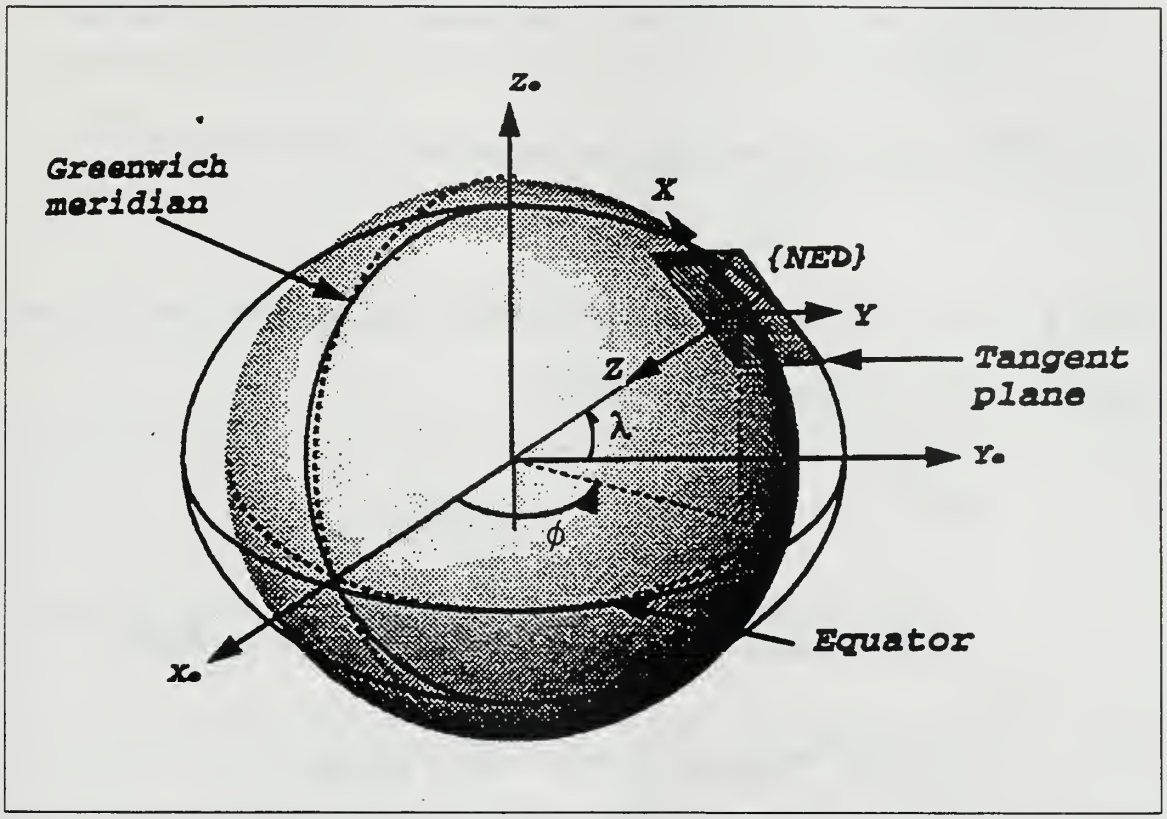

Figure 1. ECEF and Local Tangent Plane Coordinate System [Ref. 3]

4. Body Reference Frame

The aircraft body reference frame is a right hand orthogonal coordinate system with the origin at the aircraft's center of gravity. The x-axis points forward along the aircraft's longitudinal axis. The y-axis points laterally toward the right wing tip, and the z-axis points downward, normal to the x-y plane. The transformation matrix for

converting vectors (either position or velocity vectors) from universal (inertial) to body coordinate systems (${}^{B}_{U}R$) is:

$$ {}^{B}_{U}R = \begin{bmatrix} \cos\psi\cos\theta & \sin\psi\cos\theta & -\sin\theta \\ \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \sin\theta\sin\phi\sin\psi + \cos\psi\cos\phi & \cos\theta\sin\phi \\ \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi & \sin\theta\cos\phi\sin\psi - \cos\psi\sin\phi & \cos\theta\cos\phi \end{bmatrix} $$

and the transformation from body to universal coordinates is just the inverse:

$$ {}^{U}_{B}R = {}^{B}_{U}R^{-1} $$

where ϕ, θ, and ψ are the Euler angles in roll, pitch, and yaw, respectively. The body reference frame is illustrated in figure 2. [Ref. 4]

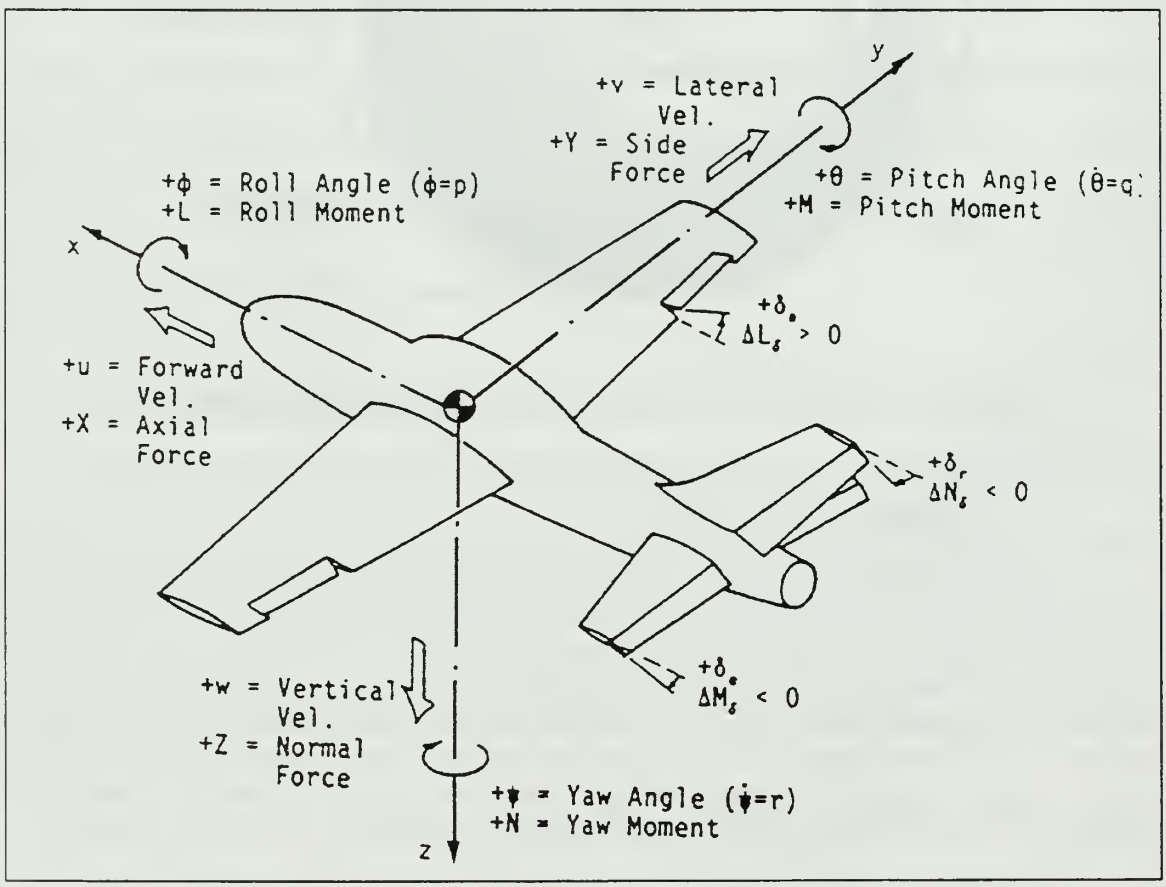

Figure 2. Aircraft Body Reference Frame [Ref. 4]

5. Camera Reference Frame

The camera reference frame is a Cartesian coordinate system with its origin located at the focal point of the camera, as shown in figure 3.

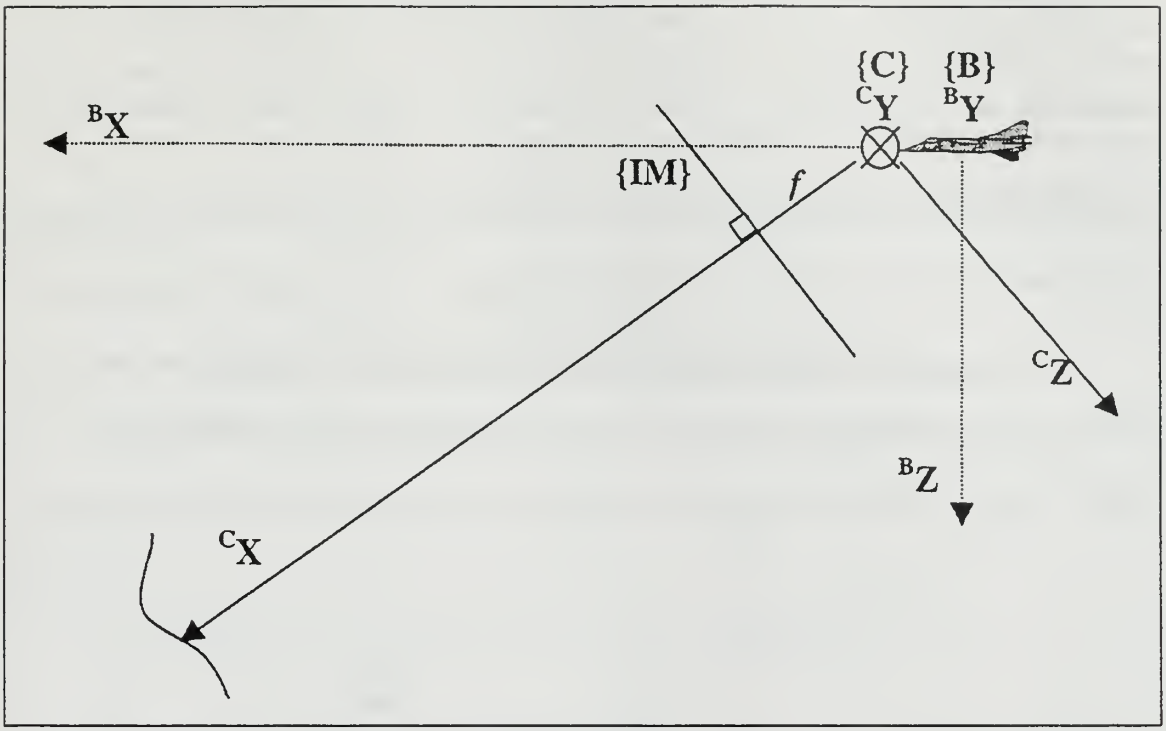

Figure 3. Camera Reference Frame [Ref. 3]

The x-axis points forward along the optical axis of the camera, and the y-axis is positive to the right (into the page as drawn). The z-axis is positive down, normal to the camera x-y plane. With the camera mounted on the aircraft, the conversion from the body reference frame to the camera reference frame would follow the same rotation sequence and, thus, use the same rotation matrix as described in the previous section. If the camera is mounted in the aircraft coincident with the aircraft body axes, then all of the Euler angles are zero, the rotation matrix is the identity matrix, and the camera reference frame equals the aircraft body axis reference frame. [Ref. 3]

6. Image Plane Reference Frame

A pinhole camera model can be used to map three-dimensional camera reference frame coordinates to two-dimensional image plane reference frame coordinates. Denote the position vector of a point, P, in the camera field of view in camera reference frame coordinates as ${}^{C}\mathbf{P}_{PC} = [x, y, z]^T$. Let f be the focal length of the camera, and let $[u, v]^T$ denote the projection of ${}^{C}\mathbf{P}_{PC}$ onto the image plane. Then

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{f}{x} \begin{bmatrix} y \\ z \end{bmatrix}.$$

The image plane reference frame is illustrated in figure 4. [Ref. 3]

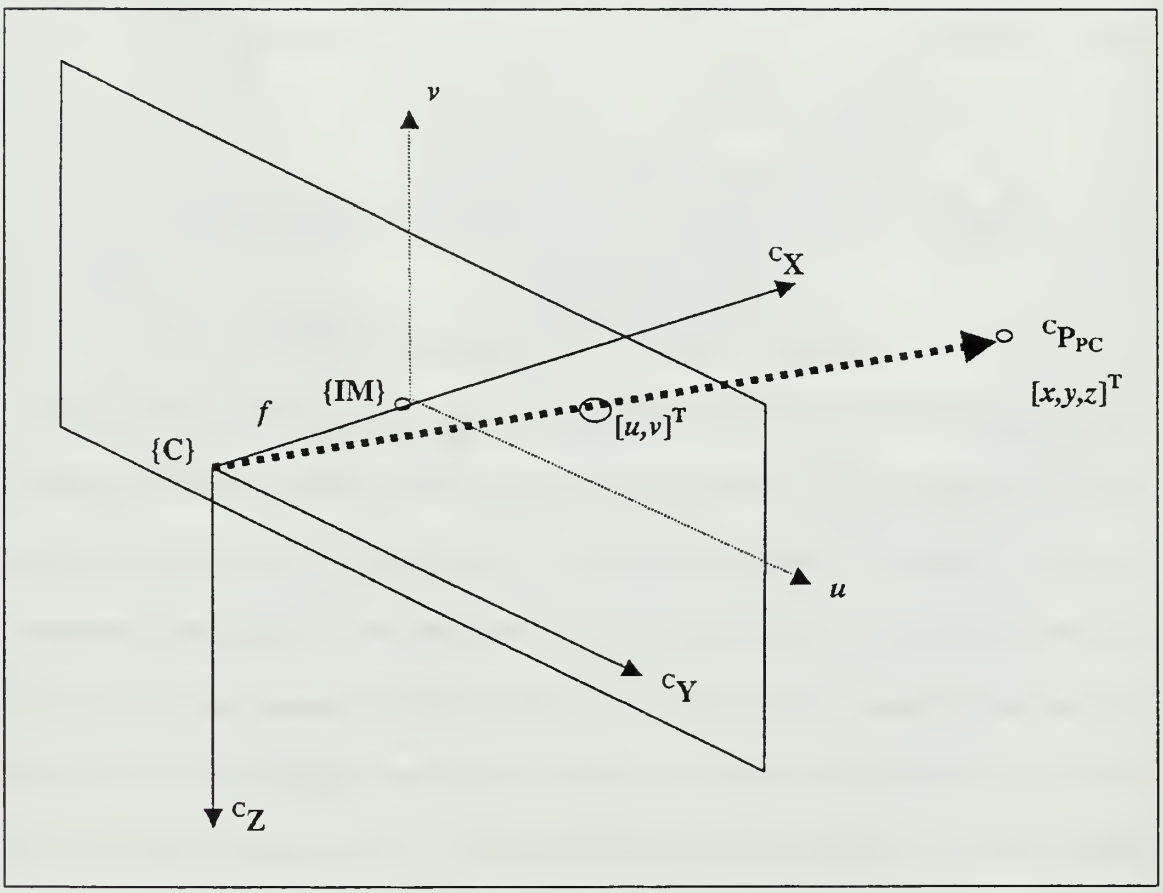

Figure 4. Image Plane Reference Frame [Ref. 3]

III. THREE POINT POSE ESTIMATION PROBLEM

A. PROBLEM FORMULATION

Given the perspective projection of three points constituting the vertices of a known triangle in three dimensional space, it is possible to determine the position of each of the vertices. Image formation in photography and human vision takes place by means of straight rays from the points of the viewed object passing through a common point and being captured by light sensitive material. This common point is called the perspective center and corresponds to the lens in the camera and eye [Ref. 6].

The problem was first formulated in 1841 by German mathematician Grunert, and can be set up in the following way, as described below and illustrated in figure 5. [Ref. 5]

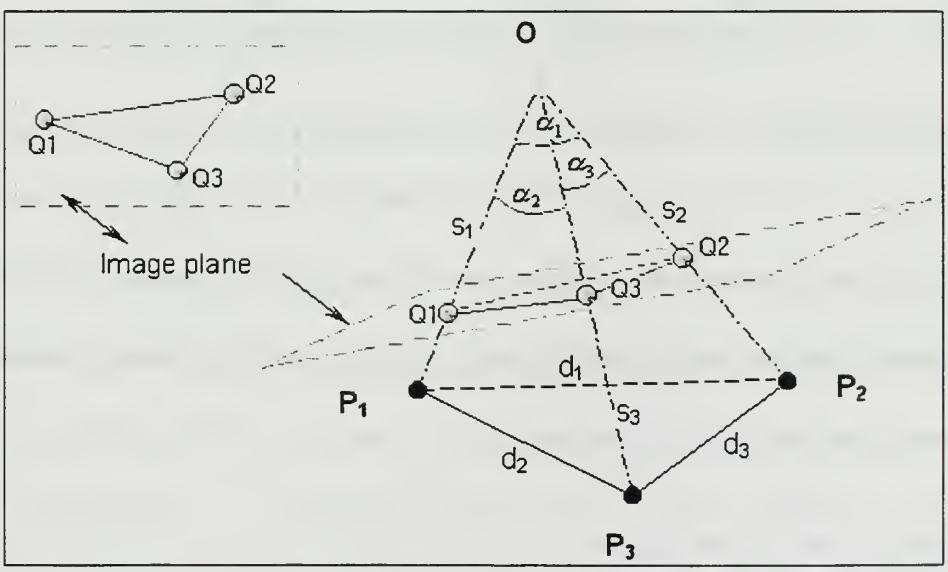

Figure 5. Geometry of the Three Point Pose Estimation Problem [Ref. 1]

Let the unknown positions of the three points of the known triangle be P_1, P_2, and P_3, and take the origin of the camera coordinate frame, O, to be the center of perspectivity. The

image projection plane is taken to be distance f in front of O. Define the vectors $\vec{p}_i = \{x_i, y_i, z_i\}$, $i = 1,2,3$ as the vectors connecting O to the three know points, P_i, $i = 1,2,3$. Let the known side lengths of the triangle be defined as:

$$\begin{aligned} d_1 &= \|\vec{p}_1 - \vec{p}_2\| \neq 0 \\ d_2 &= \|\vec{p}_1 - \vec{p}_3\| \neq 0 \\ d_3 &= \|\vec{p}_2 - \vec{p}_3\| \neq 0 \\ \text{where } & d_1 \neq d_2 \neq d_3 \end{aligned} \tag{1}$$

Let the observed perspective projection of P_1, P_2, and P_3 be Q_1, Q_2, and Q_3, respectively. The projection of each point on to the image plane has the following form:

$$Q_i = \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \frac{f}{x_i} \begin{pmatrix} y_i \\ z_i \end{pmatrix} \tag{2}$$

The points O, P_1, P_2, and P_3 form a tetrahedron. Let the angles at O opposite the sides $\overline{P_1P_2}$, $\overline{P_1P_3}$, and $\overline{P_2P_3}$ be given by α_1, α_2, and α_3, respectively. Finally, let the unknown distances of the points P_1, P_2, and P_3 from O be defined as

$$s_i = \|\vec{p}_i\|, \; i = 1,2,3. \tag{3}$$

To determine the position of the three points with respect to the camera reference frame, it is sufficient to determine s_1, s_2, and s_3 since $\vec{p}_i = s_i \vec{j}_i$, $i = 1,2,3$ where $\vec{j}_i$ represents the unit vector between O and P_i. [Refs 1 and 5]

B. GRUNERT'S SOLUTION

Using the problem formulation stated above, Grunert proceeded in the following way [Ref. 5]. Using the law of cosines, we get

$$s_1^2 + s_2^2 - 2s_1 s_2 \cos\alpha_1 = d_1^2 \tag{4}$$
$$s_1^2 + s_3^2 - 2s_1 s_3 \cos\alpha_2 = d_2^2 \tag{5}$$
$$s_2^2 + s_3^2 - 2s_2 s_3 \cos\alpha_3 = d_3^2 \tag{6}$$

Let $s_2 = us_1$ and $s_3 = vs_1$ for some u and v. Then,

$$s_1^2 = \frac{d_1^2}{1+u^2-2u\cos\alpha_1} = \frac{d_2^2}{1+v^2-2v\cos\alpha_2} = \frac{d_3^2}{u^2+v^2-2uv\cos\alpha_3} \tag{7}$$

from which

$$u^2 + \frac{d_2^2 - d_3^2}{d_2^2}v^2 - 2uv\cos\alpha_3 + \frac{2d_3^2}{d_2^2}v\cos\alpha_2 - \frac{d_3^2}{d_2^2} = 0 \tag{8}$$

$$u^2 - \frac{d_1^2}{d_2^2}v^2 + 2v\frac{d_1^2}{d_2^2}\cos\beta - 2u\cos\alpha_1^2 + \frac{d_2^2 - d_1^2}{d_2^2} = 0 \tag{9}$$

From (8)

$$u^2 = -\frac{d_2^2 - d_3^2}{d_2^2}v^2 + 2uv\cos\alpha_3 - \frac{2d_3^2}{d_2^2}v\cos\alpha_2 + \frac{d_3^2}{d_2^2} \tag{10}$$

Substituting (10) into (9) yields an expression for u in terms of v. This expression for u is then substituted into (8) to obtain a fourth order polynomial in v.

$$A_4 v^4 + A_3 v^3 + A_2 v^2 + A_1 v^1 + A_0 = 0 \tag{11}$$

This fourth order polynomial equation can have as many as four real roots. [Ref. 5]

Since Grunert first formulated his solution, this problem has been addressed by many mathematicians and scientists throughout the world. Haralick presents and compares several of the solutions at length [Ref. 5]. However, Yakimenko and Kaminer contend that none of the published solutions attempt to determine the number of

geometrically feasible solutions, nor do they show how to obtain at least one of them reliably [Ref. 1].

C. NUMERICAL THREE-POINT ALGORITHM

1. Problem Formulation

Yakimenko and Kaminer build on Grunert's problem formulation by combining (1) and (2) to obtain nine equations in nine unknowns, $\{x_i, y_i, z_i\}$, $i = 1,2,3$. Defining $\bar{a} \equiv af^{-1}$, the following comes from (2):

$$y_i = x_i \bar{u}_i, \; z_i = x_i \bar{v}_i \tag{12}$$

Substituting these expressions into (1), (1) and (2) are reduced to the following set of three non-linear equations in three unknowns:

$$\begin{aligned}
&\left(1+\bar{u}_1^2+\bar{v}_1^2\right)x_1^2 - 2\left(1+\bar{u}_1\bar{u}_2+\bar{v}_1\bar{v}_2\right)x_1x_2 + \left(1+\bar{u}_2^2+\bar{v}_2^2\right)x_2^2 = d_1^2 \\
&\left(1+\bar{u}_1^2+\bar{v}_1^2\right)x_1^2 - 2\left(1+\bar{u}_1\bar{u}_3+\bar{v}_1\bar{v}_3\right)x_1x_3 + \left(1+\bar{u}_3^2+\bar{v}_3^2\right)x_3^2 = d_2^2 \\
&\left(1+\bar{u}_2^2+v_2^2\right)x_2^2 - 2\left(1+\bar{u}_2\bar{u}_3+\bar{v}_2\bar{v}_3\right)x_1x_3 + \left(1+\bar{u}_3^2+\bar{v}_3^2\right)x_3^2 = d_3^2
\end{aligned} \tag{13}$$

To simplify notation, (13) is rewritten as

$$\begin{aligned}
&Ax_1^2 - 2D_{12}x_1x_2 + Bx_2^2 = \bar{d}_1 \\
&Ax_1^2 - 2D_{13}x_1x_3 + Cx_3^2 = \bar{d}_2 \\
&Bx_2^2 - 2D_{23}x_1x_3 + Cx_3^2 = \bar{d}_3
\end{aligned} \tag{14}$$

Note that $A, B, C, \bar{d}_i$ are strictly positive by construction. The solution of (14), shown in figure 6, is an intersection of three elliptic cylinders, whose axes of symmetry are given by x_i, where $i = 1, 2, 3$.

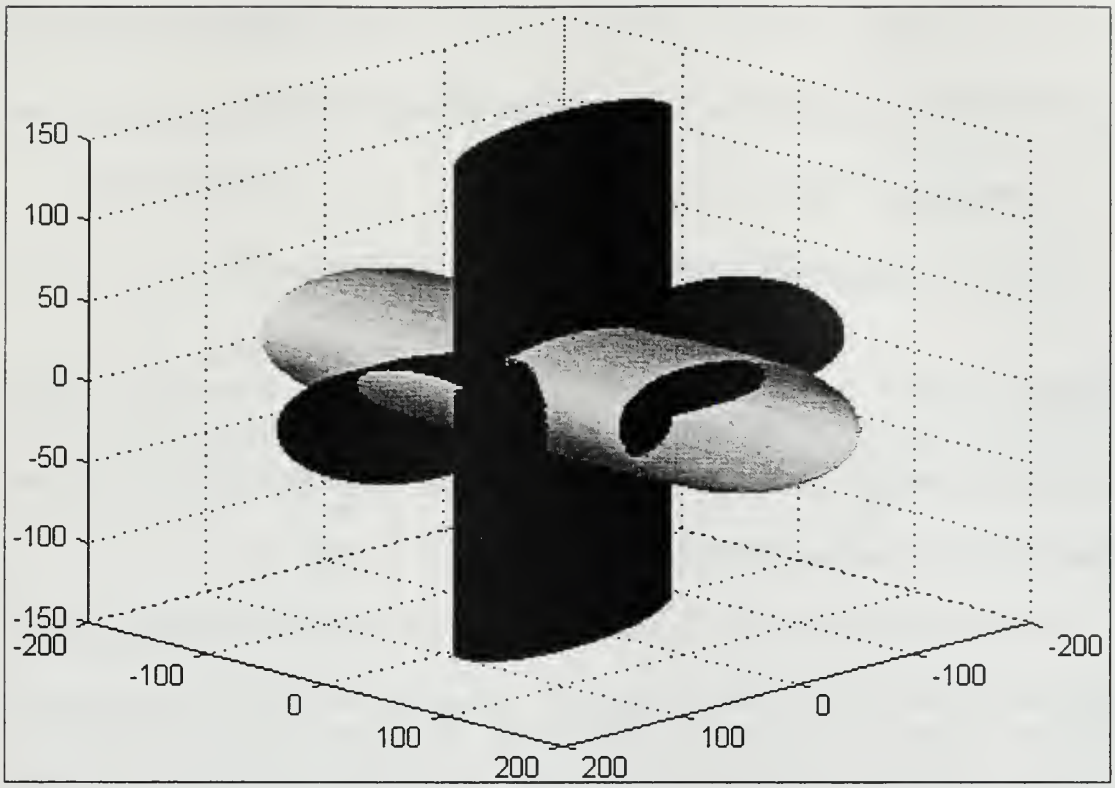

Figure 6. Intersection of 3 Elliptic Cylinders to Yield 8 Solutions [Ref. 1]

While the intersection of three elliptic cylinders may yield as many as eight solutions, Yakimenko and Kaminer show that the number of sets of admissible solutions is reduced to four if it is assumed that the camera is always in front of the landing area. By using the first two equations in (14), the following expressions for x_2 and x_3 are produced:

$$x_2 = (D_{12}x_1 \pm \sqrt{(D_{12}x_1)^2 - (Ax_1^2 - \bar{d}_1)B})B^{-1}$$
$$x_3 = (D_{13}x_1 \pm \sqrt{(D_{13}x_1)^2 - (Ax_1^2 - \bar{d}_2)C})C^{-1} \quad (15)$$

The set of possible admissible solutions x_1 lies in the interval

$$0 < x_1 \leq \min\left\{\frac{\bar{d}_1 B}{AB - D_{12}^2}; \frac{\bar{d}_2 C}{AC - D_{13}^2}\right\} \quad (16)$$

By taking all possible combinations for x_2 and x_3 and substituting them into the third equation of (14), four equations in x_1 are obtained. Denote them as Δ_{++}, Δ_{-+}, Δ_{+-}, and Δ_{--}. By setting each of these expressions to zero,

$$\Delta_{++}(x_1)=0,\ \Delta_{-+}(x_1)=0,\ \Delta_{+-}(x_1)=0,\ \Delta_{--}(x_1)=0 \tag{17}$$

admissible solutions for x_1 are obtained. Furthermore, they demonstrate through simulation using realistic values for $\bar{d}_1,\bar{d}_2,\bar{d}_3$ that two solution sets are produced, at least one of which is admissible. [Ref. 1]

Because the set of equations (14) may have two admissible solutions, a test to reliably determine the correct one is required. Yakimenko and Kaminer propose to use normals to resolve the ambiguity by constructing a set of three vectors for each solution and determining the respective normals to each of the planes defined by the vector sets. The normals are then used to identify the correct solution. [Ref. 1]

2. Numerical Algorithm

Based on the results presented in the previous section, Yakimenko and Kaminer propose the following algorithm for solving the three point problem. Suppose a good initial guess of $\hat{\bar{p}}_i^{(0)}=\{x_i,y_i,z_i\}$, $i=\overline{1,3}$ is available. Compute the normal to the plane generated by $\hat{\bar{p}}_i^{(0)}$, $i=\overline{1,3}$:

$$\bar{n}^{(0)}=\frac{(\hat{\bar{p}}_1^{(0)}-\hat{\bar{p}}_2^{(0)})\times(\hat{\bar{p}}_1^{(0)}-\hat{\bar{p}}_3^{(0)})}{\left\|\hat{\bar{p}}_1^{(0)}-\hat{\bar{p}}_2^{(0)}\right\|\left\|\hat{\bar{p}}_1^{(0)}-\hat{\bar{p}}_3^{(0)}\right\|}. \tag{18}$$

Then, for step k:

1) solve numerically equations (17) for $x_1^{(k)}$ in the interval (16), using $x_1^{(k-1)}$ as an initial guess;

2) substitute each solution $x_1^{(k)}$ obtained in 1) into (12) to get $\hat{\bar{p}}_{i_1}^{(k)}$ and $\hat{\bar{p}}_{i_2}^{(k)}$;

3) compute normals

$$\vec{n}_1^{(k)} = \frac{(\hat{\bar{p}}_{1_1}^{(k)} - \hat{\bar{p}}_{2_1}^{(k)}) \times (\hat{\bar{p}}_{1_1}^{(k)} - \hat{\bar{p}}_{3_1}^{(k)})}{\left\| \hat{\bar{p}}_{1_1}^{(k)} - \hat{\bar{p}}_{2_1}^{(k)} \right\| \left\| \hat{\bar{p}}_{1_1}^{(k)} - \hat{\bar{p}}_{3_1}^{(k)} \right\|} \text{ and } \vec{n}_2^{(k)} = \frac{(\hat{\bar{p}}_{1_2}^{(k)} - \hat{\bar{p}}_{2_2}^{(k)}) \times (\hat{\bar{p}}_{1_2}^{(k)} - \hat{\bar{p}}_{3_2}^{(k)})}{\left\| \hat{\bar{p}}_{1_2}^{(k)} - \hat{\bar{p}}_{2_2}^{(k)} \right\| \left\| \hat{\bar{p}}_{1_2}^{(k)} - \hat{\bar{p}}_{3_2}^{(k)} \right\|};$$

4) choose set $\hat{\bar{p}}_{i_1}^{(k)}$, $i = \overline{1,3}$ or $\hat{\bar{p}}_{i_2}^{(k)}$, $i = \overline{1,3}$ that maximizes the dot product $\left\langle \vec{n}^{(k)}, \vec{n}^{(k-1)} \right\rangle$.

Once the correct solution is obtained, the orientation of the camera frame with respect to the plane formed by three points of the ship can be calculated as follows. Let *{3p}* denote an orthogonal coordinate system attached to the plane generated by three points; let *{c}* denote the coordinate system attached to the camera; and let ${}_{3p}^{c}R$ be the coordinate transformation from *{3p}* to *{c}*. Form three orthogonal vectors $\vec{r}_1$, $\vec{r}_2$, $\vec{r}_3$ using the correct solution $\hat{\bar{p}}_1$, $\hat{\bar{p}}_2$, $\hat{\bar{p}}_3$ as follows:

$$\vec{r}_1 = \frac{(\hat{\bar{p}}_2 - \hat{\bar{p}}_1)}{\left\| \hat{\bar{p}}_2 - \hat{\bar{p}}_1 \right\|}, \quad \vec{r}_3 = \frac{(\hat{\bar{p}}_2 - \hat{\bar{p}}_1) \times (\hat{\bar{p}}_3 - \hat{\bar{p}}_1)}{\left\| \hat{\bar{p}}_2 - \hat{\bar{p}}_1 \right\| \left\| \hat{\bar{p}}_3 - \hat{\bar{p}}_1 \right\|}, \quad \vec{r}_2 = \vec{r}_3 \times \vec{r}_1. \tag{19}$$

Then ${}_{3p}^{c}R = \begin{bmatrix} \vec{r}_1 & \vec{r}_2 & \vec{r}_3 \end{bmatrix}$. The transformation matrix ${}_{3p}^{c}R$ can also be expressed using Euler angles. From this, yaw, pitch and bank angles can be found in the following manner:

$$\psi_{3p} = \tan^{-1}\frac{r_{12}}{r_{11}},\ \theta_{3p} = -\sin^{-1} r_{13},\ \phi_{3p} = \tan^{-1}\frac{r_{33}}{r_{32}}. \tag{20}$$

Now, the attitude of *{c}* with respect to a coordinate system *{s}* attached to the ship can be found using (12) from the transformation matrix ${}^{c}_{3p}R\,{}^{3p}_{s}R$, where ${}^{3p}_{s}R$ can be obtained from the known positions of the three points on the ship in *{s}*. [Ref. 1]

IV. IMAGE PROCESSING

A. PURPOSE

The intent of the image processing portion of this investigation is to establish a methodology for locating three spots of interest in a sequence of successive still images that comprise a video clip, live or recorded. These three spots represent three visibly significant points that could be seen on a moving ship at sea serving as a landing platform for a UAV.

B. METHODOLGY

A digital image is comprised of multiple rows of picture elements, commonly referred to as pixels. A pixel is the smallest addressable point of a bitmapped screen that can be independently assigned color and intensity. A bitmap is a digital representation of a picture in which all the pixels comprising the picture are rendered in a rectangular grid and correspond to specifically assigned bits in computer memory [Ref. 7]. This rectangular organization of ordered, real-valued image data makes a digital image ideally suited for representation using an array. Monochrome images can be stored in two-dimensional arrays in which each element in the array corresponds to a single pixel intensity, or luminance, in the displayed image. Color images can be stored in three-dimensional arrays, where each plane in the third dimension represents the pixel intensities for one of the three primary colors: red, green, and blue [Ref. 8]. This investigation is concerned primarily with monochrome images, presumably created by an infrared (IR) sensor .

In a digital image pixels are indexed sequentially in rows from top to bottom and in columns from left to right, which is consistent with standard mathematical notation for indexing two-dimensional arrays, or matrices. The location of each pixel in the image can be described by a vector, **p**, from the origin at the upper left hand corner of the image to the pixel, represented by row and column position coordinates i and j, respectively, in a rectangular coordinate system, such that $\mathbf{p} = i\mathbf{i} + j\mathbf{j}$. The row and column indices of the image array correspond directly to the magnitudes of the component vectors of **p**.

Pixel resolution, the number of pixels per unit length of image, is typically such that features of interest within the image are usually comprised of multiple pixels. Of interest in this investigation is the location of three distinct groups of pixels that correspond to known physical features represented by the image. For the purpose of this investigation, a spot is defined as a group of pixels whose luminance significantly exceeds that of their surroundings and that are located within a specified vertical and horizontal difference between successive pixels, δ_i and δ_j, respectively.

Determination of the image plane coordinates of the centers of three spots without a priori characterization of their nature proved difficult in the first image of the sequence. Several methods, each with its own limitations, were developed and are described below. In each case, some a priori characterization of the three spots is required. A representative image is presented in figure 7.

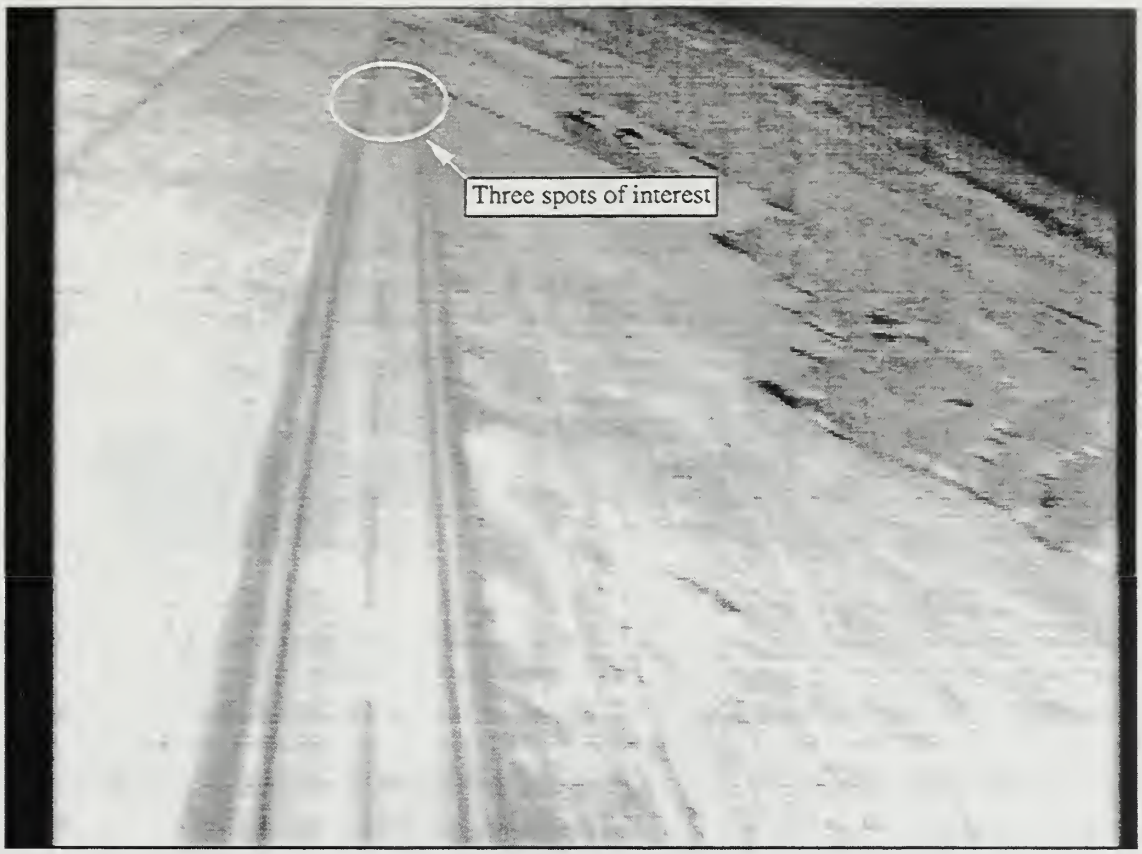

Figure 7. Unfiltered 'picture25.bmp'

1. Bisection Thresholding

On a contour plot of the image where the contours represent varying pixel luminance, the three spots of interest appear as three-dimensional spikes, as in figure 8.

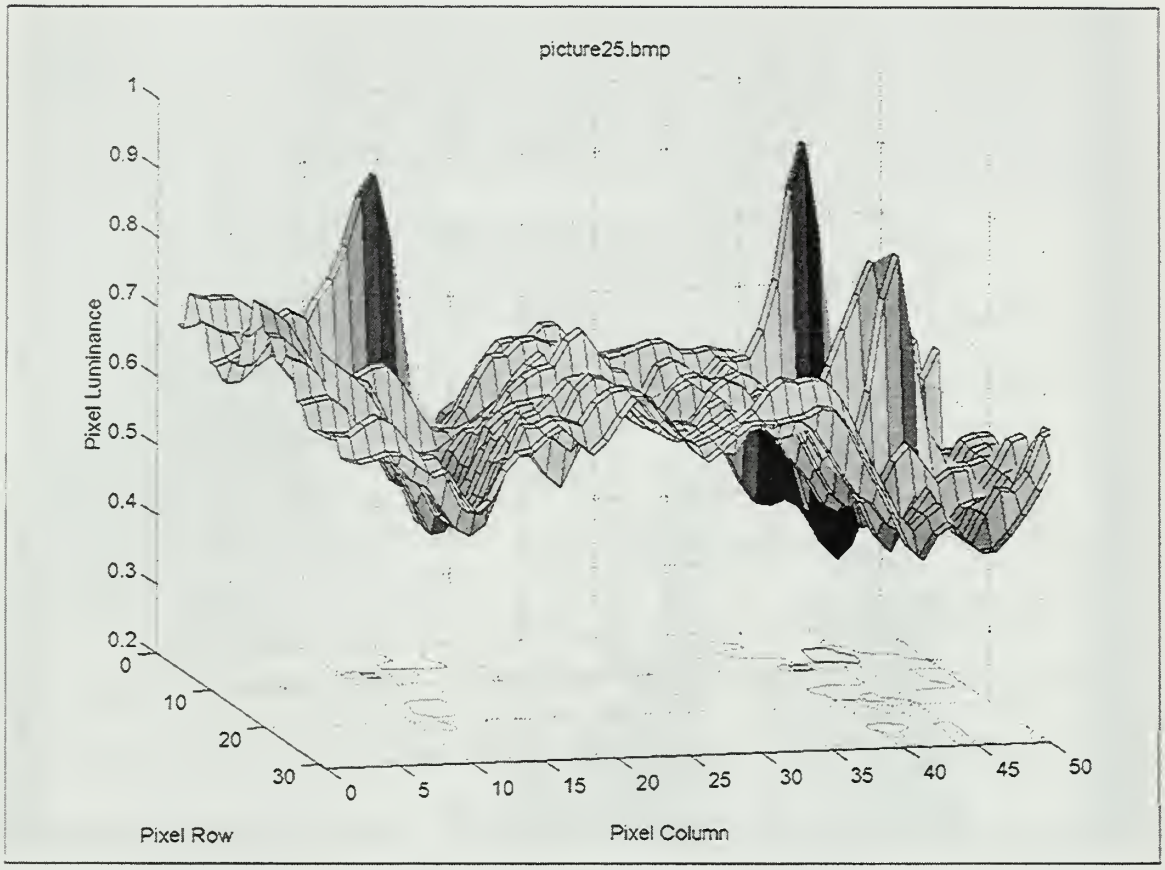

Figure 8. Pixel Luminance Contour of Three Spots of Interest and Surroundings

One method of isolating the three spots of interest from their immediate surroundings is to apply a threshold to the image, above which only the pixels whose luminance exceeds the threshold are discernable. This effectively results in slicing off the tops of the peaks of the contour plots. Assuming that the luminances of each of the spots of interest exceed the luminance of their surroundings, a threshold can be chosen such that only pixels that comprise the spots of interest exceed the threshold. Once the appropriate threshold level is chosen, the pixels that exceed the threshold are assigned to spots based on their relative position, and the center of each of the spots is computed using a weighted average.

Because of numerous dynamic factors during the landing approach, the luminance of the three spots of interest relative to their surroundings changes continuously with each successive frame during the approach. These factors include decreasing range from the camera (aircraft) to the spots (and, therefore, changing amounts of atmospheric absorption of the energy from the spots) and continuous change of the camera gain via its automatic gain control (AGC). As a result, the luminance threshold used to isolate the spot pixels from the surrounding pixels must be determined dynamically with each image frame that is processed.

a. Bisection Thresholding

The correct threshold is determined using an algorithm that adjusts the luminance threshold by iteratively bisecting the available range of pixel luminance until exactly the desired number of spots is extracted. Following each threshold adjustment, the image is evaluated to determine how many spots are contained above the threshold level. Additionally, the available range of pixel luminance is revised based on the actual number of spots extracted at a given threshold level prior to the next iteration, as described in the algorithm below.

$T_U \equiv$ upper threshold limit
$T_L \equiv$ lower threshold limit
$T_C \equiv (T_U + T_L)/2 + T_L$ {current threshold}
$N \equiv$ desired number of spots

evaluate number of spots in image @ T_C
while the number of spots $\neq N$
    if number of spots $< N$
        set $T_U = T_C$
        recompute T_C with new T_U and same T_L
    if number of spots $> N$

set $T_L = T_C$
recompute T_C with new T_L and same T_U
evaluate number of spots in image @ new T_C
return to *while* condition

For example, pixel luminance for each image ranges from 0 to 1. Based on a characterization of data from numerous approaches, the spots are indiscernible from their surroundings if the pixel luminance of the three spots of interest is less than 0.4. Therefore, assign 0.4 as the initial lower limit of the threshold range. The upper limit of the range is initially 1. For the first image, the algorithm sets the threshold at the center of the available range, in this case, 0.7. If more than three spots were extracted at this threshold, the lower threshold limit would be set to the current threshold (0.7), and the upper limit would remain unchanged. A new threshold (0.85) is computed such that it bisects the new threshold range (0.7 to 1). If fewer than three spots were extracted at the first threshold, the upper threshold limit would be set to the current threshold (0.7), and the lower limit would remain unchanged. A new threshold (0.55) would be computed such that it bisected the new threshold range (0.4 to 0.7). This process is iterated until exactly three spots are extracted from the image.

b. Spot assignment and counting

For each threshold level evaluated, the number of spots associated with that threshold level must be determined. However, when the threshold is applied to the image, or a portion thereof, the resulting data set consists of the coordinates of the individual pixels whose luminance exceeds the threshold. Thus, to determine the number

of spots, the pixels must be grouped accordingly into their associated spots so they can be counted.

The coordinates of all the pixels whose luminance exceed the threshold are sorted in ascending row order and placed in an array hereafter referred to as the pixel array, P. The first pixel in the pixel array is moved to a new array to establish a new spot and is assigned as the basis for comparison with the other pixels in P. If the next pixel in P meets the criteria to be considered in the same spot as the first pixel, it is removed from P, appended to the spot array, and is assigned as the new basis for comparison with the next pixel in P. If it is not considered to be part of the same spot as the first pixel from P, then it is placed in a separate array for all pixels that are not in the current spot and the first pixel remains the basis for comparison with the next pixel in P. This process is repeated until all of the pixels in the pixel array have been evaluated and assigned to a spot. An algorithm that determines the number of spots at the chosen threshold operates on the pixel array and assigns each of the pixels to one of the spots, as described below.

$P \equiv$ pixel array, made up of individual pixel coordinate pairs, $\mathbf{p}$
$S_C \equiv$ array of coordinates of pixels comprising current spot
$S_H \equiv$ array of coordinates of pixels not comprising current spot
$\mathbf{p}_b \equiv$ pixel used as basis for comparison with remaining pixels in P

while $P \neq [\ ]$
    copy first pixel from P to S_C to establish new spot
    assign the first pixel from P as $\mathbf{p}_b$
    for k = 2 to number of pixels in P
        if $\{(|\mathbf{p}_{ki} - \mathbf{p}_{bi}| < \delta_i)\ \&\ (|\mathbf{p}_{kj} - \mathbf{p}_{bj}| < \delta_j)\}$
            append $\mathbf{p}_k$ to S_C
            let $\mathbf{p}_b = \mathbf{p}_k$
        else
            append $\mathbf{p}_k$ to S_H
            $\mathbf{p}_b$ remains same

return to *for* condition
copy S_C elsewhere for later use
let $S_C = [\ ]$
let $P = S_H$
return to *while* condition

At the completion of the algorithm, each of the S_C arrays generated contains the pixel data for one of the spots, and the number of S_C arrays generated equals the number of spots.

c. Spot center computation

The center of a spot, $\mathbf{p}_c$, composed of n pixels, $\mathbf{p}$, is determined by weighting the position of each pixel by its luminance, l, as represented by the following equation.

$$\mathbf{p}_c = \frac{1}{L}\sum_{p=1}^{n} l_p \mathbf{p}_p \ ; \ \text{where} \ L = \sum_{q=1}^{n} l_q$$

In figure 9 below, the pixels that exceed the threshold comprise the spots and are shown as collections of dots. Each of these three groups of pixels is considered one spot. The crosses represent the computed center of the associated spot.

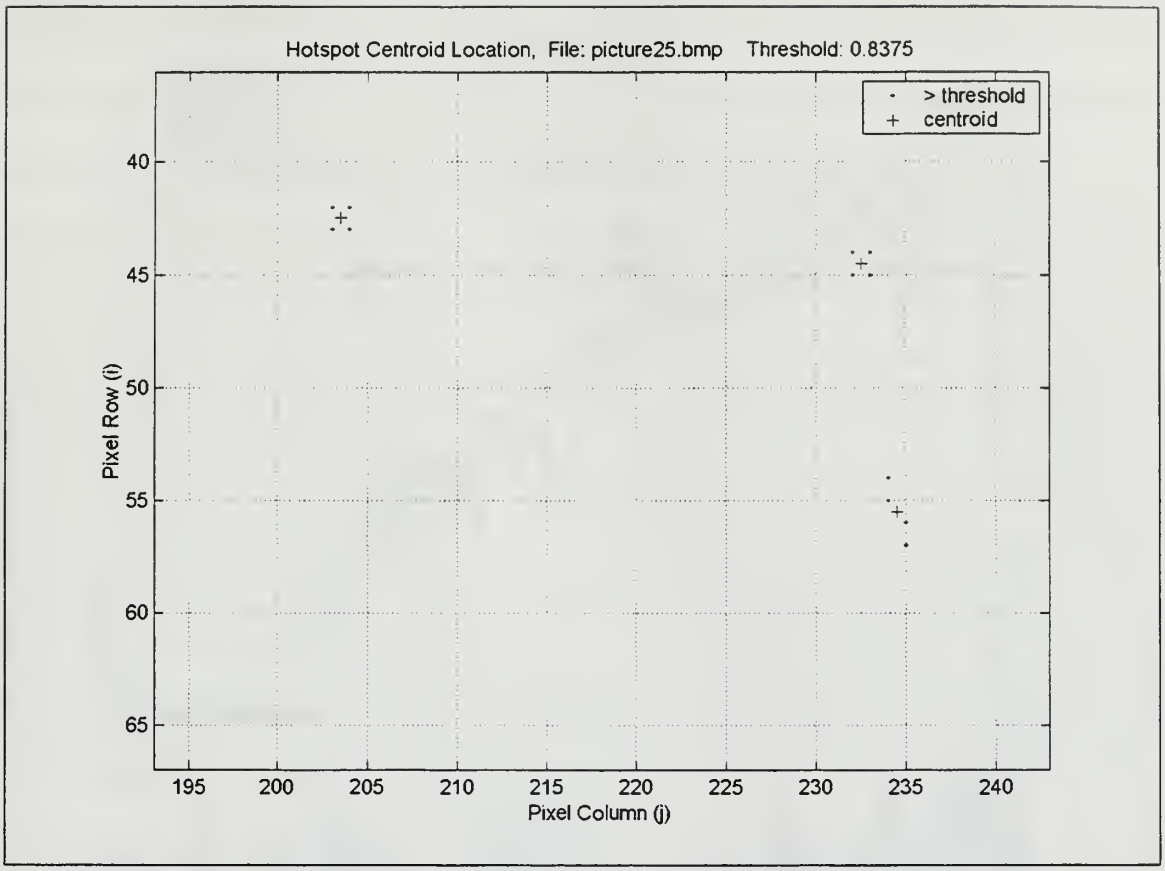

Figure 9. "Hotspot" Pixel Groupings and Center Locations

2. Polynomial-Difference

Another viable method for determining the image plane coordinates of the three spots involved analyzing the rows and columns in the image array using an algorithm described below.

For each row in the pixel luminance array, a second-order polynomial was fit to the pixel luminance data in a least squares sense. A representative plot of luminance data and its associated polynomial curve fit are presented below in figure 10 for a row that includes one of the three spots of interest. The normalized luminance for each pixel are shown as diamonds, and the solid line represents the second-order polynomial fit.

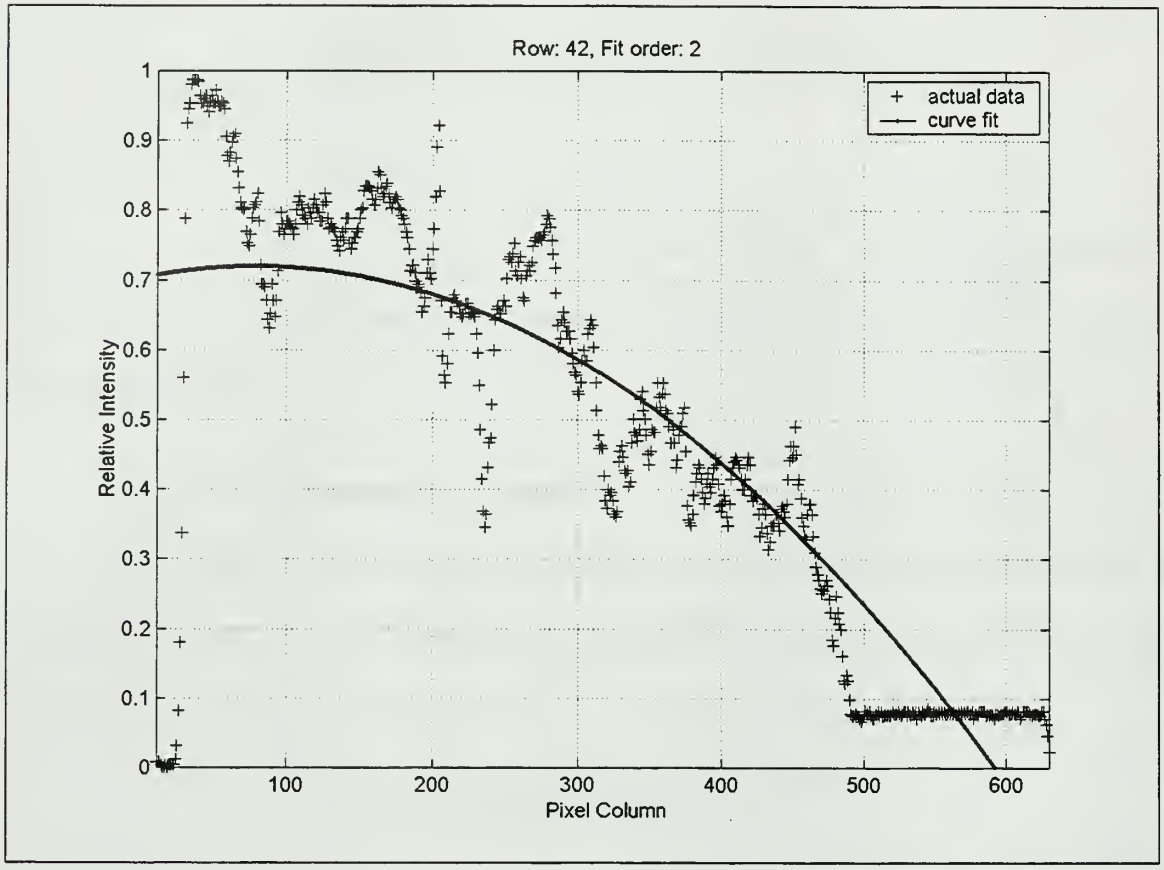

Figure 10. Representative Polynomial Fit to Pixel Luminance Data (Unfiltered)

In this case, the column position of the point of interest is 204, but it is difficult to discern by computing the difference between the actual luminance and the polynomial curve fit value for that column. It was assumed that the column at which the maximum difference between the actual luminance and the second-order polynomial fit occurred would correspond to the column position of one of the pixels that comprised one of the spots of interest. However, as can be seen in figure 10, this assumption does not hold when this polynomial fit-difference method is applied to an unfiltered image. A composite plot of all the computed difference values from the row-wise and column-wise calculations is presented in figure 11. In the image for which these data are computed,

the spots of interest are located at column positions 204, 233, and 235. It is clear from this figure that the data at column position 204 is overshadowed by data at several other locations in the image that do not correspond to the three spots of interest.

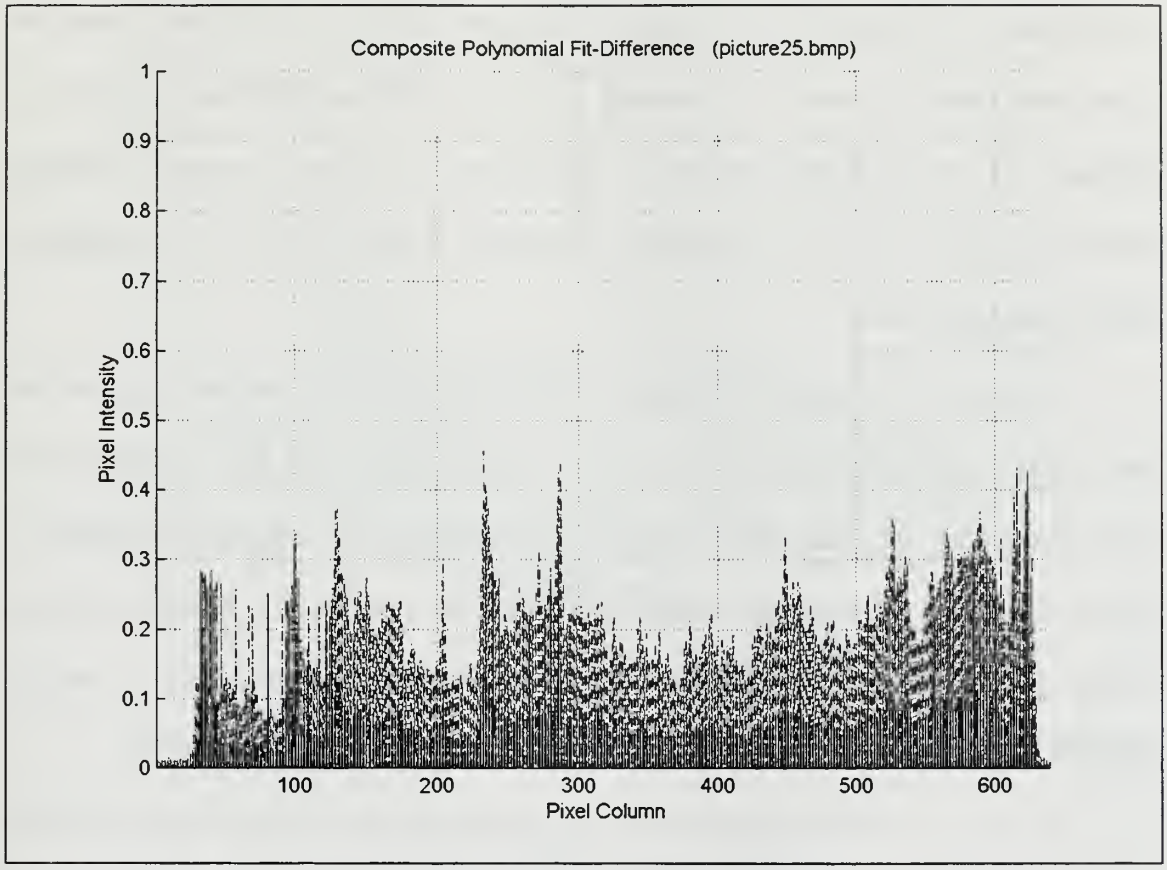

Figure 11. Composite Polynomial Fit-Difference Plot

In order to apply this method with certainty so that the three spots of interest would be found, additional information regarding the nature of the image is required.

Given that the field of view of the camera is centered on and fixed in the plane defined by the aircraft's x and z body axes, it is assumed that the three spots of interest first appear in the image frame as the aircraft completes its left-hand approach turn and is close to being lined up on the runway as it begins its landing approach. This places the

runway to the left of the center of the image frame, extending from top to bottom. Furthermore, the camera is depressed approximately 5 degrees from horizontal. Based on these assumptions, it is most likely that the three spots of interest will appear near the top of the image frame, slightly left of center. In fact, the likelihood of the vertical position of the three spots of interest is considered to follow a Weibull distribution, while the likelihood of the horizontal position is considered to follow a Normal Gaussian distribution. Empirical data are combined with these conclusions to create a composite filter as described below.

To apply the polynomial-difference method effectively, it is necessary to filter the image array in such a manner that the three spots of interest can be discerned from their local surroundings yet other pixels in the image that are not of interest but of equal or greater luminance are effectively ignored. As discussed above, the likelihood of the vertical position of the three spots of interest is considered to follow a Weibull distribution as described by the following relation.

$$\mathbf{f}_W(x_i;\lambda,\sigma_W) = \frac{\lambda}{\sigma}\left(\frac{x_i}{\sigma}\right)^{\lambda-1} e^{-\left(\frac{x_i}{\sigma}\right)^{\lambda}}$$

$$\lambda = 1.8$$
$$\sigma_W = 60$$
$$\mathbf{x}_i = [1..r]$$
$$r \equiv \textit{number of rows in the image array}$$

The resulting row filter is presented in figure 12.

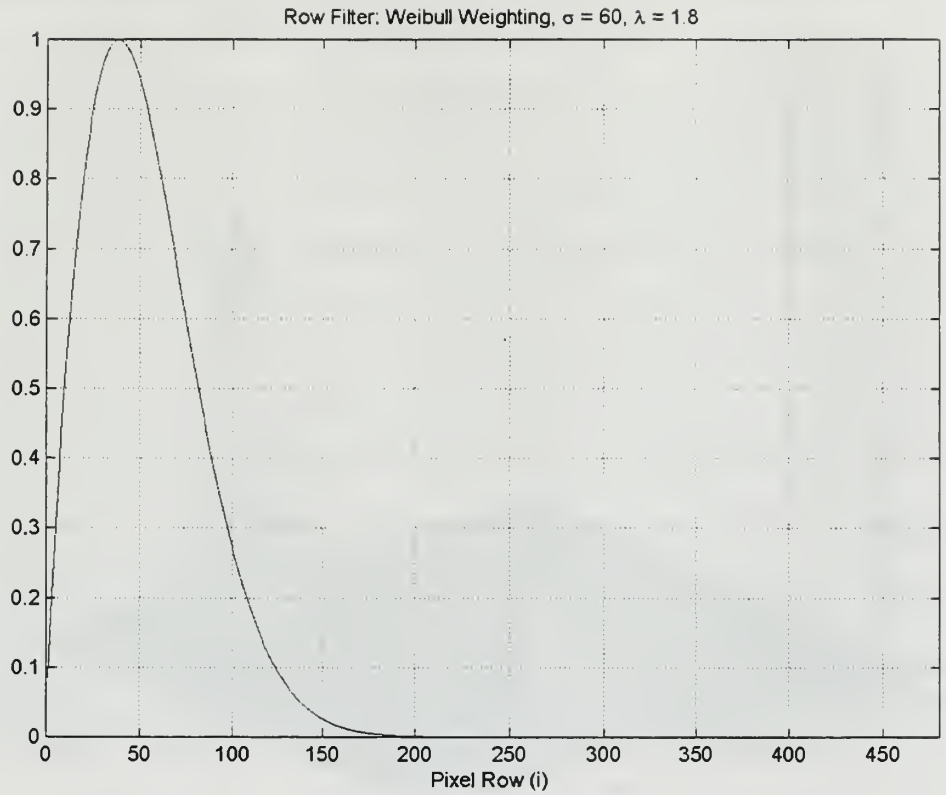

Figure 12. Weibull Row Filter

Likewise, the likelihood of the horizontal position is considered to follow a Normal Gaussian distribution as described by the following relation.

$$\mathbf{f}_N(x_j;\mu,\sigma_N) = \frac{1}{\sigma_N\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x_j-\mu}{\sigma_N}\right)^2}$$

$$\mu = 220$$
$$\sigma_N = 30$$
$$\mathbf{x}_j = [1..c]$$

c ≡ number of columns in the image array

The resulting column filter is presented in figure 13.

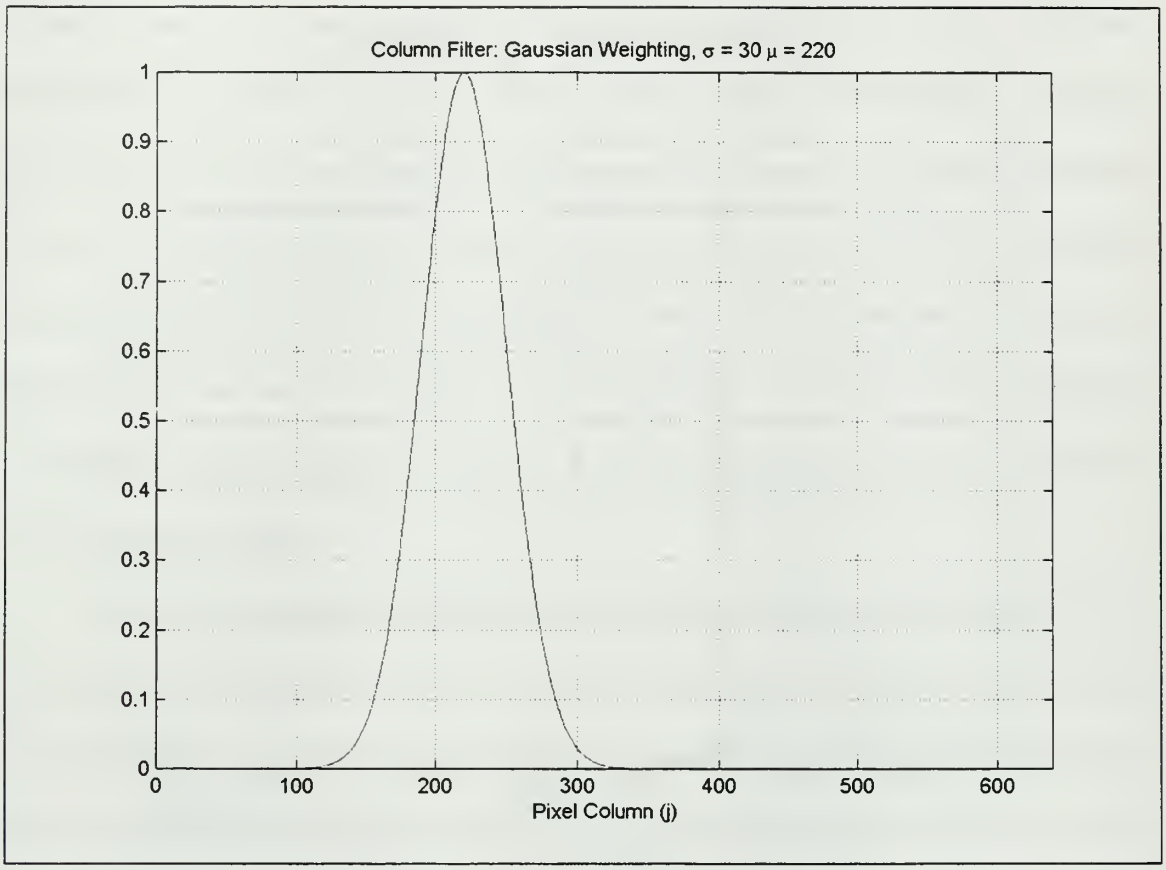

Figure 13. Normal Gaussian Column Filter

The filtered image, $\mathbf{I}_F$, is produced by premultiplying, in a matrix sense, the image array, **I**, by the row filter, **R**, and postmultiplying by the column filter, **C**, as described in the following relation.

$$\mathbf{I}_F = \mathbf{RIC}$$

where R and C are diagonal matrices as defined below:

$$\mathbf{R} = \left(diag\left(\frac{\mathbf{f}_W}{\max(\mathbf{f}_W)} \right) \right)^T$$

$$\mathbf{C} = diag\left(\frac{\mathbf{f}_N}{\max(\mathbf{f}_N)} \right)$$

The resulting composite image filter is graphically depicted in figure 14.

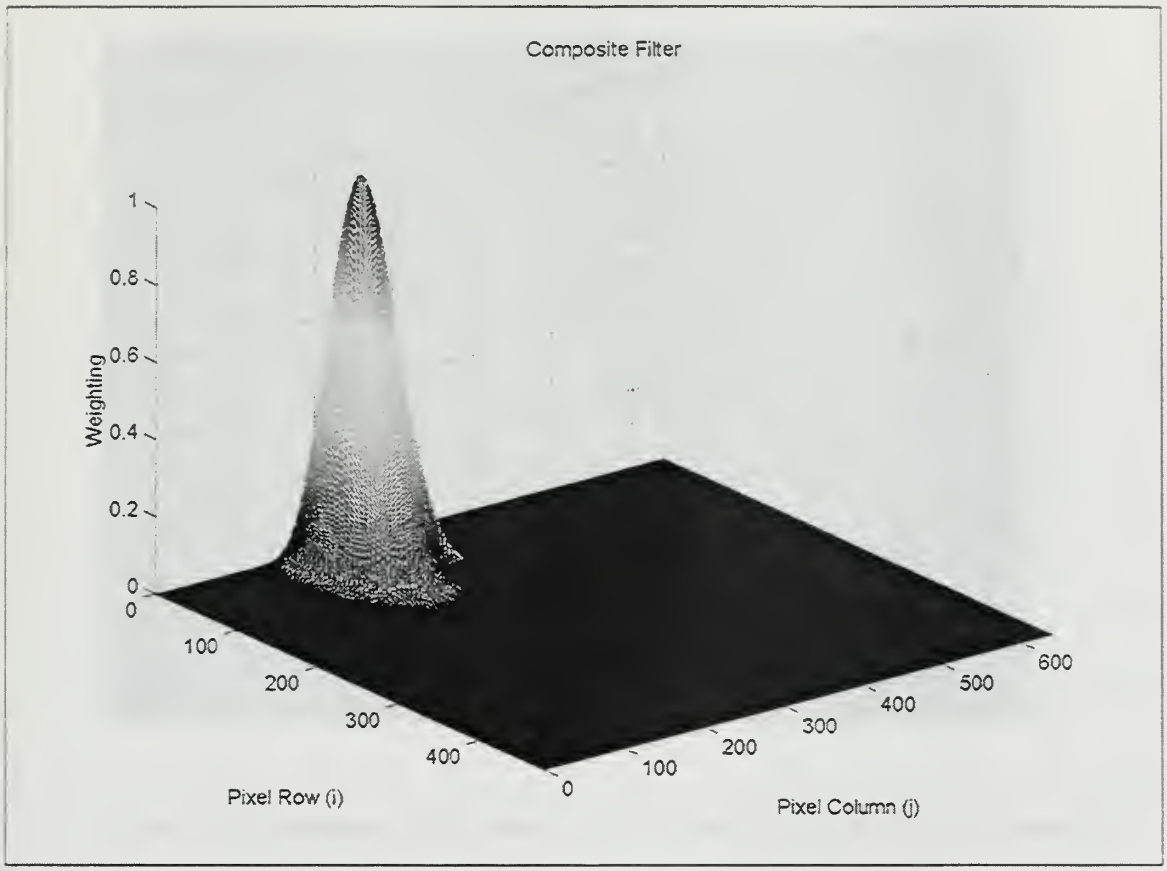

Figure 14. Composite Image Filter

The effects of applying the filter to the first image in the sequence to be processed can be seen in figure 15.

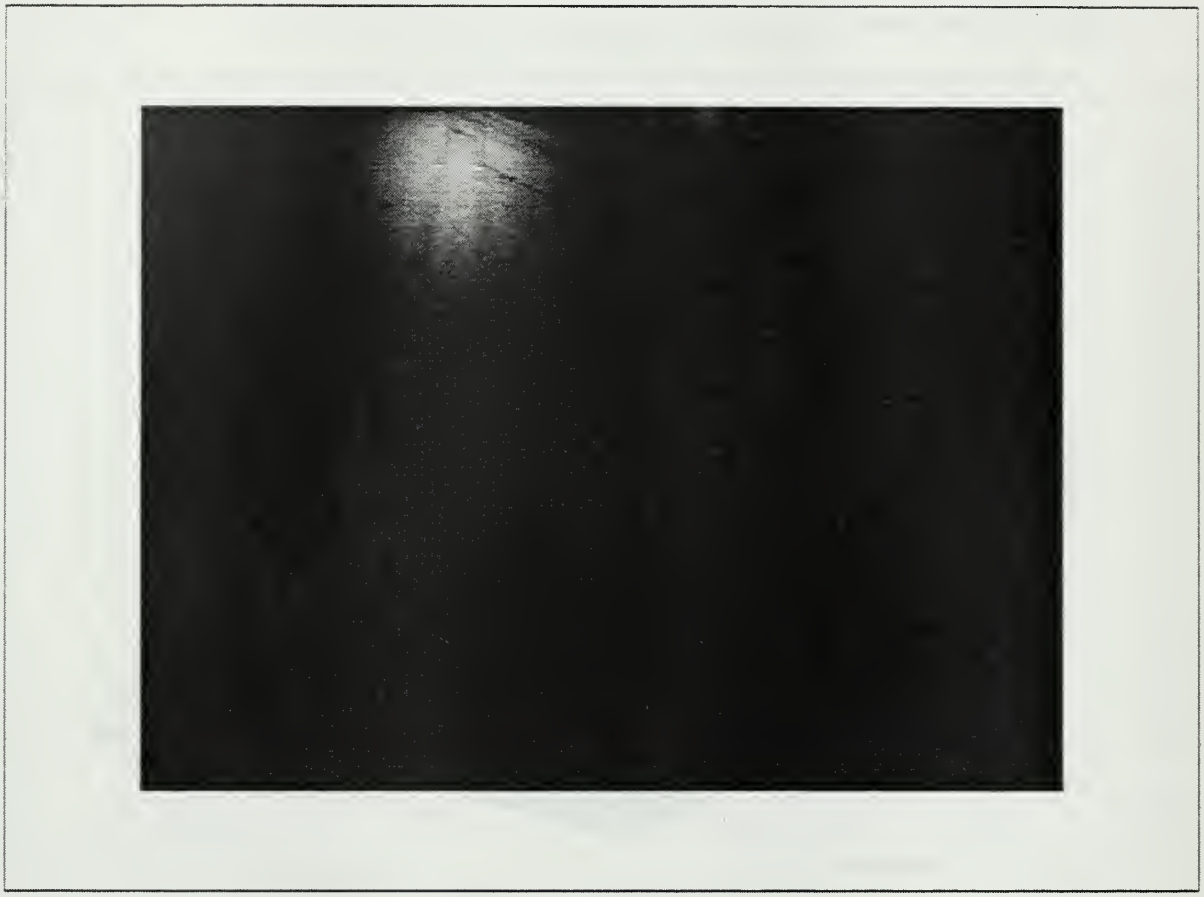

Figure 15. Filtered 'picture25.bmp'

The polynomial difference method can be effectively applied to the filtered image array to extract the desired horizontal and vertical position information of the three spots of interest. A representative polynomial fit to filtered data is provided in figure 16, which clearly shows the maximum difference between the data and the fit to occur at column 204, the location of the point of interest.

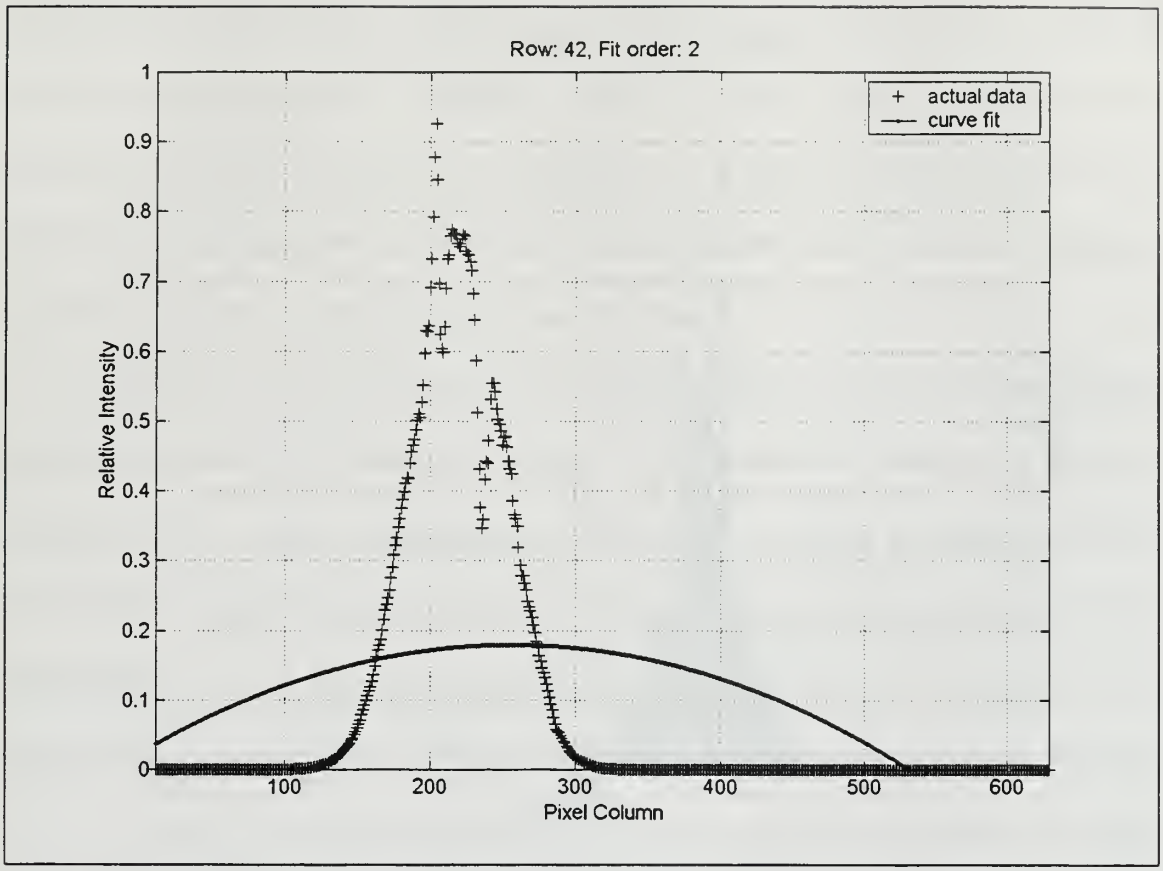

Figure 16. Representative Polynomial Fit to Pixel Luminance Data (Filtered)

This computation is repeated for each row in the image array. Since exactly one maximum difference value is determined for each row, this computation is also repeated column-wise for each column in the array to mitigate the instance when one row contained pixels for more than one of the spots of interest. As a result, a total of 1120 pixels are identified. A composite plot of pixel intensities versus pixel column position for all of the 1120 identified pixels is presented in figure 17. Note the three tallest groups of intensities correspond to the three spots of interest.

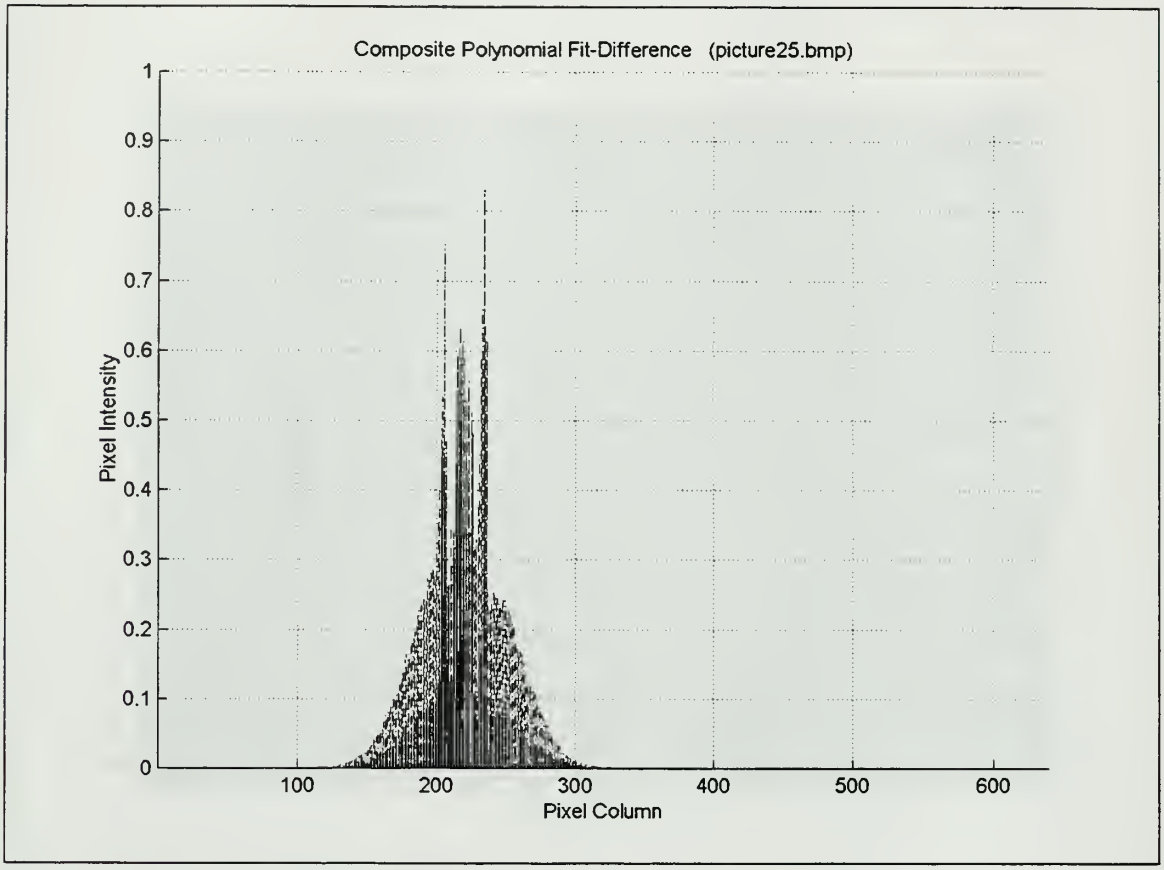

Figure 17. Pixel Intensities vs. Column Position, Polynomial Difference Method

3. Sliding Average

A third method relies only on the assumptions that the three spots of interest are among those in the image that appear significantly brighter than their immediate surroundings and that the distance between them is small relative to the size of the image to be processed. This method is effective for locating the three spots of interest in the first image of the sequence to be processed.

This method preprocesses the image array by computing the mean luminance value for each row of pixels in the image. In each row, the row mean luminance value is then subtracted from each individual pixel luminance value. The luminance value of

each pixel whose luminance value is below the row mean is set to zero. A standard sliding average procedure is applied to each row of the image. The width of the averaging window is determined empirically to satisfy the requirements of the application. Any pixels whose luminance values are greater than ten standard deviations above the mean are identified; all others are set to zero.

The next step in this approach is to determine which of the remaining non-zero pixel are adjacent, and therefore, comprise a single spot and which ones represent different spots. This is accomplished by locating the brightest remaining pixel and setting all of the pixels within specified horizontal and vertical distances of it to zero. The magnitudes of these horizontal and vertical distances are also a function of the application. This procedure is iteratively applied to the group of remaining non-zero pixels until all of the non-zero pixels have either been marked as a spot center or zeroed because of their proximity to another bright pixel.

Once all of the bright spots in the image have been located, geometry of the spots of interest is used to determine which of the bright spots are the three spots of interest. In this application of this method, it is reliably assumed, based on a characterization of the images to be processed, that the three spots of interest will be closer to each other than any other bright spots identified in the image, especially at the ranges that are typical for initial visual acquisition of the landing area. As such, the distances between all of the bright spots in the image are computed, and are summed three at a time. The combination of distances that yields the smallest sum identifies the three points of interest.

C. COMPARISON OF METHODS

Because information gained from processing one image can be used to simplify the processing of the successive image in a sequence of still images that comprise a video representation of an aircraft approach to landing, the task of determining the image plane coordinates of the three spots of interest in each image can be divided into two portions: 1) determining the coordinates in the first image of the sequence, and, 2) determining the coordinates in each of the remaining images. Each of the methods discussed above is differently suited to both of these tasks.

1. Initial Coordinate Determination

Based on analysis of numerous images from the beginning of several landing approach video clips, the images from the initial portion of the approach can be generally characterized as follows. The three spots of interest are located in the top third of the image, slightly left of center. They are brighter than their immediate surroundings, but they are not necessarily the three brightest spots in the entire image. The three spots are located within approximately 35 pixels of each other horizontally and 15 pixels vertically, and each spot is less than 4 pixels wide and less than 6 pixels tall.

a. Bisection Thresholding

As discussed previously, the bisection thresholding method is based on the assumption that the three spots of interest are the three brightest spots in the portion of the image that is being analyzed. Because it can not be assumed that the three spots of interest are the three brightest spots in the entire image, the portion of the image to be analyzed must be reduced to a point where any spots that are brighter than the three spots

of interest are eliminated. This is accomplished by placing an appropriately sized search box around the area where the three spots of interest are expected to appear. To give reasonable assurance that the search box will not include any spots brighter than the three of interest, it is typically not sized much larger than the area in which the three spots are located. This fairly precise sizing of the search box necessitates precise placement of it as well if this method is to be successful. However, because only general assumptions can be made about the location of the three spots of interest in the first image to be analyzed, this method is not suitable for reliably determining the image plane coordinates of the three spots in the first image.

b. Polynomial Difference

The precision with which the filter must be positioned using polynomial-difference method is not as stringent as that with which the search box is placed in the binomial threshold method. However, to most effectively shape and position the respective Weibull and Gaussian filters appropriately to properly isolate the three spots of interest requires that the three spots appear in the first image reasonably close to where they are expected. As long as the three spots consistently appear close to the same position within the image at the beginning of the image sequence, this method will produce acceptable results. However, if this consistency can not be achieved, the shape and position parameters for the distributions need to be changed accordingly. This is a potentially tedious prospect, depending on how inconsistently the three spots are located in each initial image.

c. Sliding Average

Unlike the previous two methods, the sliding average does not depend on any *a priori* knowledge of the position of the three spots of interest. Instead, it relies on the three spots being significantly brighter than their immediate surroundings, being among the brightest several spots in the entire image, and being located relatively close to each other compared to the other bright spots in the image. All of these three dependencies are satisfied by the scenario under consideration. Thus the sliding average method is most suitable for determining the image plane coordinates of the three spots of interest in the initial image of the sequence.

2. Successive Coordinate Determination

Whereas the dependence on *a priori* knowledge of the three points' position effectively eliminated the polynomial difference and bisection thresholding methods from consideration for analysis of the initial image in the sequence, all three of the methods are legitimate candidates for processing the successive images in the sequence. Clearly, when the video clip of a landing approach consists of approximately 480 still images gathered at a rate of 30 images per second, the time required to process each image is of great interest. For the sake of comparison, a MATLAB script was developed for the sliding average and polynomial difference methods that ran each process iteratively on the same image 140 times. The image was the initial image in a representative video clip recorded during an approach to landing made during actual flight test of a UAV. To generate the bisection thresholding data, the method was applied to the first 140 successive images in a representative video clip of an approach performed during flight

test. Each of these scripts was run on the same personal computer (PC), which was configured with a 600 MHz Central Processing Unit (CPU), 256 MB of Random Access Memory (RAM), and the Microsoft Windows NT Version 4.0 operating system. During the time that each of these scripts was running, no other user application was in use; that is, all available resources within the PC were available to the script alone. This evaluation facilitated an order of magnitude comparison of the three proposed methods.

a. Sliding Average

While the sliding average method's generality make it best suited for processing the initial image in the sequence, that generality results in it being less desirable for processing the successive images in the sequence. To retain its generality, this method performs numerous computations on each pixel in the image, which equates to 307,200 pixels in a 640 by 480 image. The average time required to perform the sliding average method on a single image from early in the landing approach is approximately 34.9 seconds, as determined via the test described in the previous paragraph. The results of this test are presented as a histogram in figure 18.

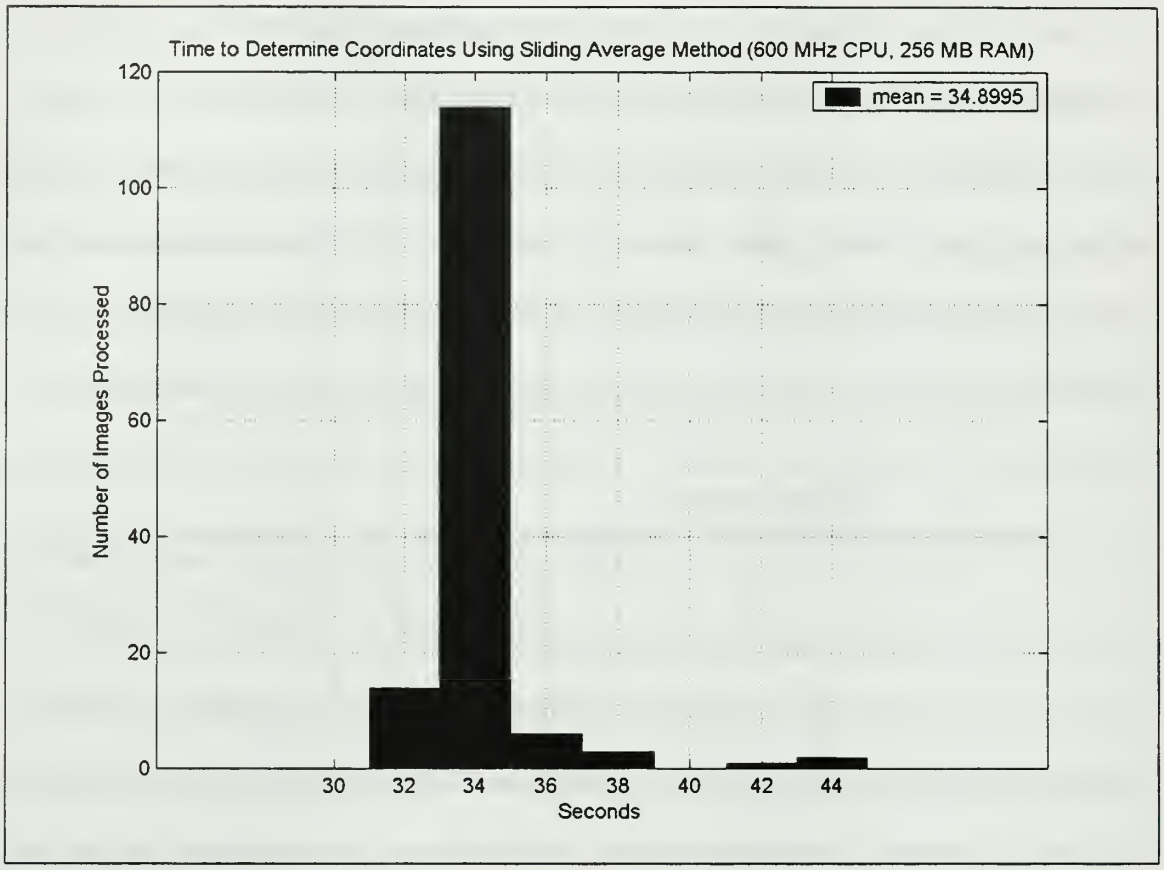

Figure 18. Sliding Average Method Performance

An additional weakness of the sliding average method is its assumption that the three spots are closer to each other than any of the other bright spots in the image. While this is generally true in the early portion of the landing approach, as the aircraft gets closer to the landing point, the three spots spread apart in the image plane until they are the full width of the image (640 pixels) apart. The resulting deterioration in the method's performance in processing images from the latter half of the landing approach make it unacceptable for processing the image sequence.

b. Polynomial Difference

The polynomial difference method was also qualitatively evaluated in the manner described above. As can be seen in the histogram presented in figure 19, this method enjoys an order of magnitude improvement in speed over the sliding average method under the test conditions described. The average time required to perform the polynomial difference method on a single image from early in the landing approach is approximately 4.1 seconds.

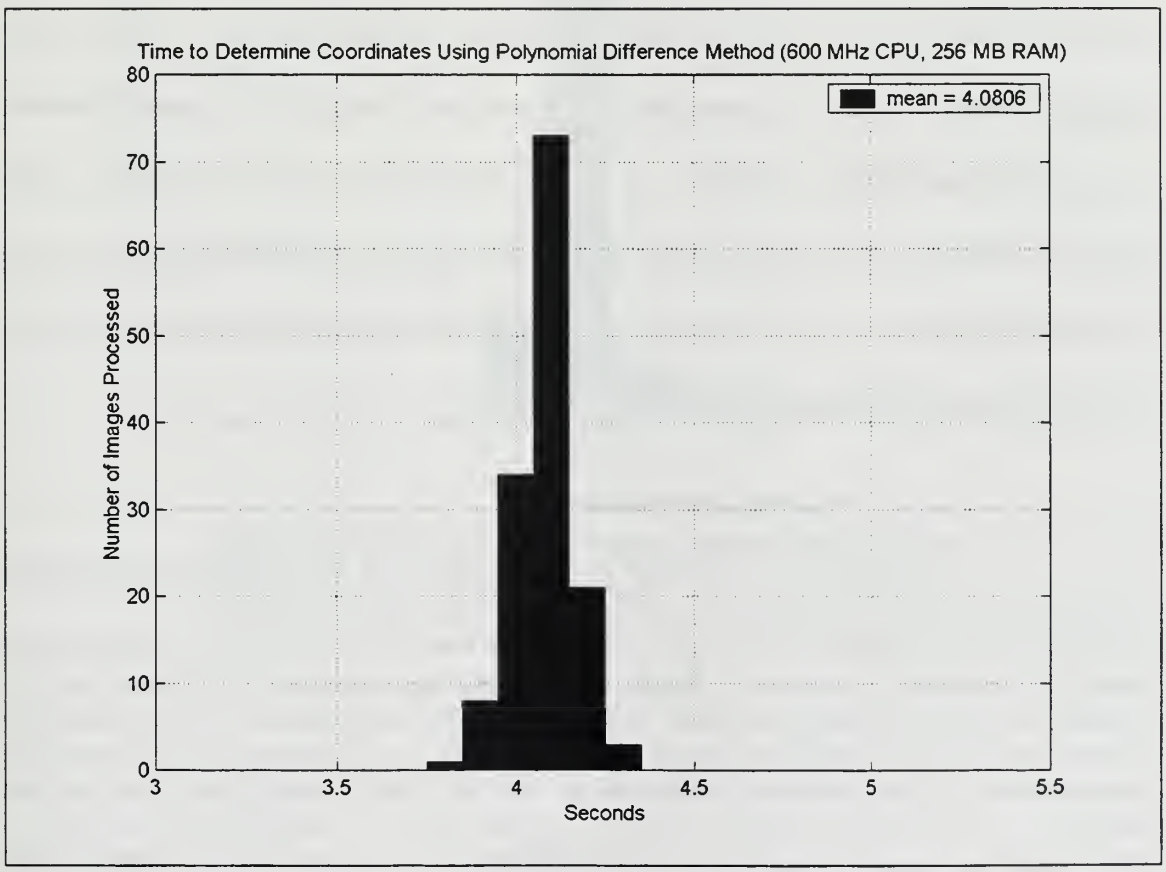

Figure 19. Polynomial Difference Method Performance

The major weakness in the polynomial difference method with respect to processing the successive images in the sequence is related to the shape and location

parameters of the two distributions that comprise it. As the landing approach proceeds, the three spots of interest migrate apart in the image plane, and the center of the triangle they define could also potentially move within the image. It is this triangle center that is centered in the filter weighting scheme as previously described. In order to maintain the center of the weighting filter over the center of the spot triangle, both the shape and location parameters for each of the distributions would need to be adjusted and a new filter generated with each image processed to ensure the resulting filtered image could be reliably thresholded to produce the correct coordinates of the three spots. Because this approach would be more computationally involved than computing the spot coordinates in a single image using a static filter, it is clear that the mean time to process a single image in a sequence of successive images using the polynomial difference method would increase significantly from the results presented above although no simulation was performed to substantiate this hypothesis.

c. Bisection Thresholding

As mentioned previously, to be effective the bisection thresholding method must be applied to an image, or portion thereof, in which the three spots of interest are the three brightest spots in the area under consideration. To evaluate the performance of the bisection thresholding method, each image had a rectangular processing box placed around the area in which the three spots of interest were expected to appear under the assumption that any spots in the image brighter than the three of interest would be located outside the processing box. The bisection thresholding method was then applied to the area inside the processing box for each image in the video clip

under the test conditions described above. The results of this evaluation are presented in figure 20.

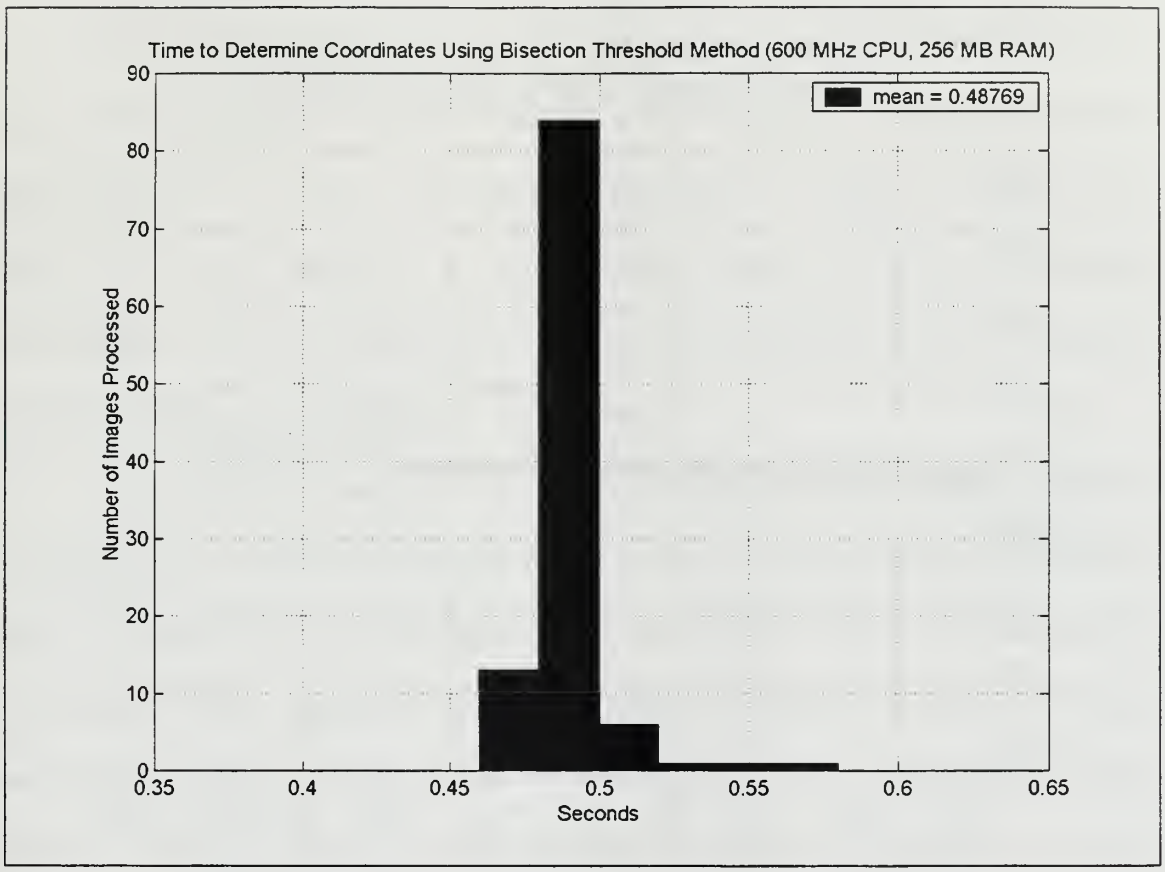

Figure 20. Bisection Thresholding Method Performance

With an average time to determine the coordinates of the three spots of approximately 0.5 seconds, the bisection thresholding enjoys an order of magnitude performance improvement over the polynomial difference method and a two order of magnitude improvement over the sliding average method under the test conditions described previously. Furthermore, the bisection thresholding method was applied to a full length video clip of an entire approach to landing consisting of approximately 480 successive images. The longest time required to determine the coordinates of the spots in

any of the images was less than 0.9 seconds, and the vast majority of the images were processed in approximately 0.5 seconds. These results are presented in figure 21.

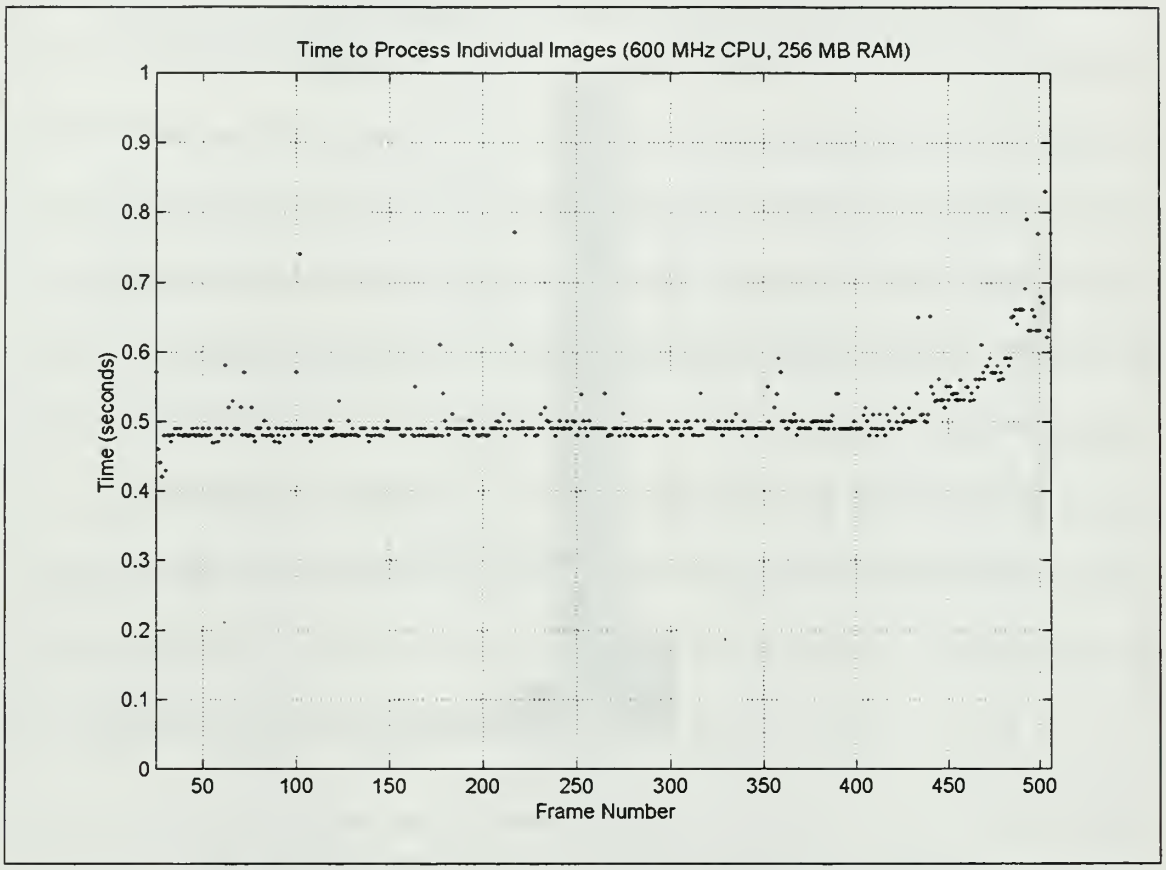

Figure 21. Time Required to Process Each Image in Landing Approach Video Clip

Clearly, when it is possible to predict the approximate location of the spots in the subsequent image frame based on the spot coordinates of the current frame, the bisection thresholding method is the most efficient of the three methods presented.

D. IMPLEMENTATION

1. Introduction

The algorithms described previously in this chapter are implemented in MATLAB functions and combined to process onboard video recordings of numerous landing approaches flown by a UAV. Because these algorithms are intended ultimately to be used to process flight imagery in real-time, emphasis is placed on implementations that would process each image expeditiously. It should not be forgotten, however, that this is a prototype implementation with room for refinement.

The algorithms are designed to process recordings captured on a digital video tape recorder from an infrared video camera placed in the nose of an aircraft. In accordance with the National Television Standards Committee (NTSC) standard, the frame rate of the captured video is assumed to be approximately 30 frames per second [Ref. 9], so each second of video can be parsed into 30 still images that can then be processed using algorithms described above. Because the typical approach is approximately 15 to 17 seconds long, analysis of each approach requires between 450 and 510 individual image files to be processed. The software used to process the sequences of still images is implemented in locally written MATLAB Version 5.3 and SIMULINK 3 functions. The native functions in the Image Processing Toolbox are especially useful. The capabilities of the Image Processing Toolbox are completely described in the User's Guide [Ref. 8], and are summarized below as required. The code for all of the locally written algorithms and functions is provided in Appendix A.

The individual still images are processed offline to determine the image plane coordinates of the three spots of interest in each frame. The image processing algorithms are implemented to process images at as high a rate as possible to support a high position estimate update rate based on vision sensors. A sliding average algorithm is applied to the first image of the sequence to locate the three spots of interest because this algorithm has the least dependence on *a priori* knowledge of the spot characteristics, as described previously. Once the three spots are located in the first image of the sequence, their positions are used to predict the location of the three spots of interest in the subsequent image. Based on this prediction, a rectangular processing box is sized and positioned over the next image in the sequence to include the three spots of interest but to exclude as much of the remainder of the image as possible. A bisection thresholding algorithm is then applied to the portion of the image that is contained within the processing box to determine the image plane coordinates of the three spots of interest. Once the spot coordinates are determined for the current image frame, they are used to adjust the rectangular processing box dimensions and position to include the predicted position of the three spots of interest in the next frame. This process is repeated until the coordinates of the three spots of interest are determined in all of the still images in the sequence. The collection of coordinates of these three spots are used as the input to an implementation of Yakimenko and Kaminer's numerical three-point algorithm that computes estimates of the aircraft's position and velocity in the Local Tangent Plane.

2. Image Capture and Storage

The video tape recorder (VTR) used to capture the live IR camera video of the approaches is also used to play the recorded video back in NTSC standard format for post-flight analysis. The VTR is interfaced to a Windows NT™ personal computer (PC) via the IEEE 1394 interface standard using an Adaptec AHA-8945 Combined 1394/Wide Ultra SCSI PCI Host Adapter. The associated Adaptec software utility, *DVDeck*, permits control of VTR functions from the PC. *DVDeck* is used to play the recorded approaches and store them to the PC hard drive. The approaches can be stored either as movie files in .avi format or as sequences of individual still images, or frames, which are stored in Microsoft Windows Bitmap (.bmp) format, with a pixel resolution of 640 x 480. For analysis of each recorded approach of interest, the recorded video is converted to a sequence of still frames which are saved as individual truecolor (RGB) .bmp files on the PC hard drive.

3. Program Structure

The MATLAB-based program written to determine the image plane coordinates of the three spots of interest is divided into numerous subroutines that are defined along logical functional boundaries. Each of the subroutines performs smaller, well-defined tasks in support of the main program. The main program, *controller.m*, is used to direct the sequence of and conditions under which the image processing subroutines are called, to define and initialize global variables and constants, and to call data analysis subroutines.

4. Image Loading and Normalization

MATLAB supports several graphics file formats, including the Microsoft Windows Bitmap (.bmp). When an RGB image is read from a .bmp file using the MATLAB native *imread.m* function, the image data is assigned to an m-by-n-by-3 array , where m is the number of pixel rows and n is the number of pixel columns. Each of the levels in the third dimension of the image array represents the data for one of the three colors. The values in the arrays represent individual pixel luminance and are stored as unsigned 8-bit integers ranging from 0 to 255. Although the .bmp file contains RGB data, in this application it represents a black and white IR camera image. After the file is loaded into the workspace, it is converted from an RGB image to grayscale by the native *rgb2gray.m* function, which eliminates the hue and saturation information while retaining the luminance [Ref. 8]. This makes it possible to keep all relevant image information in a two dimensional array rather than a three-level, three-dimensional array, thereby reducing by two-thirds the size of the array required to represent the image. After the original RGB image array was converted to a grayscale image array, the values of the array were converted to double precision integers and divided by 255 to normalize the luminance values to range between 0 and 1 using the locally written *imnorm.m* function. This array is hereafter referred to as the image array. This operation facilitates subsequent image processing algorithms without losing any fidelity from the original image file.

5. First Image

The first image is loaded into the MATLAB workspace and normalized as described above. The sliding average algorithm is applied to the image array for the first image in the landing sequence. It is expected that the three spots are brighter than their immediate surroundings, but the only assumption that can be made with respect to their position is that they initially appear somewhere in the upper third of the image. Without a more specific estimate of the spots' initial position, the Polynomial-Difference method can not be applied effectively.

A local function, *averagebox.m*, applies the sliding average algorithm to the first image of the sequence of images to be processed in order to initially determine the coordinates of the three spots of interest. Once the spot coordinates are determined, the initial processing box size and position are determined using the local function *nextbox.m*.

a. Sliding Average

The image array passed to *averagebox.m* is processed one row at a time, starting with the bottom row of the image and proceeding sequentially to the top of the image. First, the average pixel luminance value of the row is computed, then each pixel of the row with a luminance less than the row average is set to zero. For each pixel with a luminance value greater than the row average, its luminance value is replaced by the difference between the luminance value for that pixel and the row average. A seven-pixel window is then "slid," one pixel at a time, over all the pixels in the row. At each

position of the window, the intensity value of the pixel on which the window was centered is replaced by a quantity computed as follows:

$$l_{center_1} = l_{center_0} - \frac{\sum_{i=1}^{2n+1} l_i}{n+1}$$

$n \equiv$ number of pixels in the window on either side of window center

All negative elements of the row are then replaced with zeros. Positive elements of the row are unchanged by this operation. The mean luminance, $\bar{l}$, and standard deviation, σ, of the row are then calculated. Each element of the row is then replaced with either the quantity $(l - \bar{l} - 10\sigma)$ or 0, whichever is greater. This effectively reduces to zero all elements in the row except those that are extremely bright relative to their surroundings. This process is repeated for each row of the image array. A representative image with the location of each of these bright spots as determined by the algorithm superimposed on it as '+' symbols is presented in figure 22. Note the number of '+' symbols over the three spots of interest as well above and left of the landing area and along the right side of the runway near the bottom of the image.

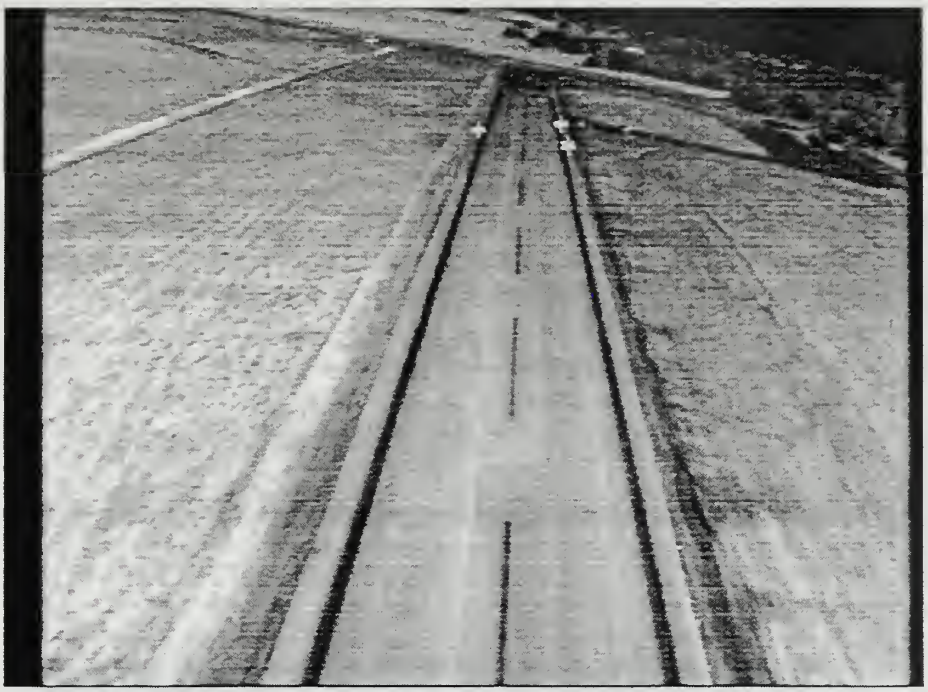

Figure 22. Representative Processed Image with Brightest Spots Indicated

After the image array is manipulated as described above, the five brightest groups of pixels (i.e., spots) in the image are located as described below. The new, manipulated image array is searched for the brightest pixel in the entire array. Once located, a square measuring 15 pixels on each side is centered over it, and the luminance values of all other pixels within the square are set to zero. This prevents the algorithm from locating five bright pixels that are adjacent to each other (i.e., part of the same spot) because the intent of this algorithm is to locate the five brightest *groups* of pixels (i.e., spots) and not just the five brightest pixels. The size of this square is determined empirically by observing the size and orientation of the spots of interest in numerous

images from the initial phase of the landing approach when the landing area is first visible. If the squares were too small, they would not include all of the pixels that comprise a given spot, resulting in one spot being represented as two. If the square were too large, it may contain more than one spot of interest, thereby masking two spots as one. The coordinates of the center of the square are noted, and this procedure is repeated until the five brightest pixels groups are identified. A representative image with the location of the five brightest spots as determined by the algorithm marked by white squares is presented in figure 23. Note white squares around each of the three spots of interest and two spots on the right edge of the runway near the bottom of the image.

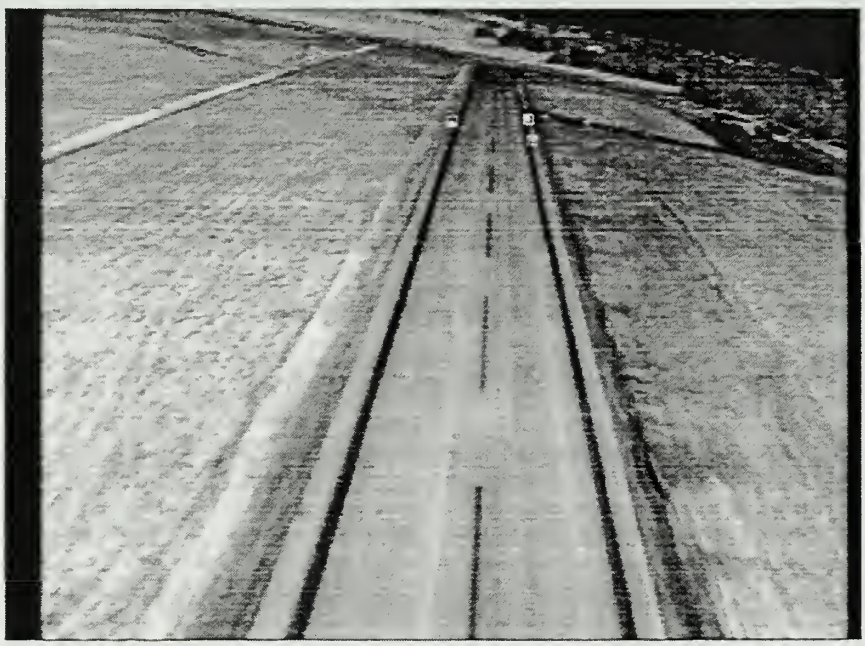

Figure 23. Representative Image with Five Brightest Spots Indicated

The next task for the algorithm is to identify which three of the five pixel groups represent the three spots of interest. This is done by computing the distance from each spot to the other four and adding the distances. The three spots with the smallest distance sums are considered to be the three spots of interest. This method is successful when the three spots of interest are located relatively close to each other, and there are no other spots of similar luminance in their immediate vicinity. No *a priori* knowledge of their location in the image is required. A representative image with the location of the three spots of interest as determined by the algorithm superimposed on it as 'o' symbols is presented in figure 24. Note that each of the 'o' symbols is over one of the spots of interest.

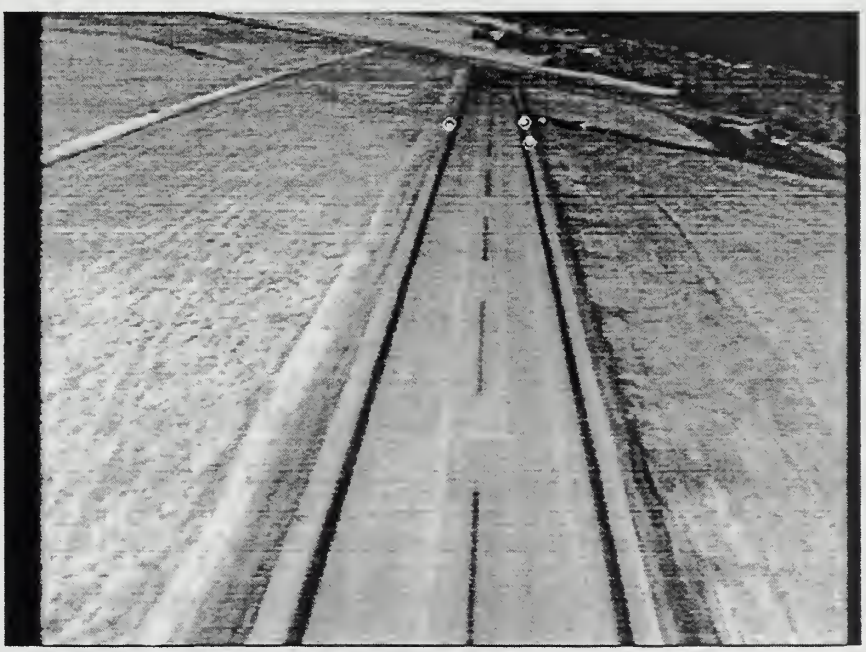

Figure 24. Representative Image with the Three Spots of Interest Indicated

Once the image plane ($[u,v]$) coordinates of the three spots of interest are determined, they are transformed to $[i,j]$ coordinates and passed to the local function *nextbox.m* to generate a processing box for the first image of the sequence to be processed.

6. Successive Images

Because only $1/30^{th}$ of a second lapses between frames, it is anticipated that the positions of the three spots of interest will not change drastically from frame to frame. Furthermore, with the IR camera effectively boresighted on the three spots through the entire approach, it was also anticipated that there will be very little vertical or horizontal drift of the group of spots across the image frame, other than that caused by minor pitch and heading adjustments and aircraft vibration. Instead, the three spots are expected to gradually spread away from each other in the image plane until the camera is close enough that all three spots can not be contained in the image frame simultaneously. However, because the camera boresight is fixed relative to the aircraft body axes, large vertical and horizontal drift rates of the spots in the image plane are induced by sudden pitch and yaw maneuvers of the aircraft during the approach. Recognizing the incremental changes in spot position, it is expected that a processing box can be successfully positioned to capture the three spots of interest from frame to frame based on the position of the spots in the current frame.

a. Processing Box Size and Position Determination

A local function, *nextbox.m*, determines the size of and predicts the location for the processing box for the subsequent image once the image plane

coordinates of the three spots of interest are determined. This function determines the difference between the largest and smallest *i* coordinates of the three spots of interest, Δ_i, and the difference between the largest and smallest *j* coordinates of the three spots of interest, Δ_j. The top boundary of the subsequent processing box is determined by subtracting the product of Δ_i and a scaling factor from the smallest *i* coordinates of the three spots of interest. The bottom boundary of the subsequent processing box is determined by adding the product of Δ_i and the same scaling factor to the largest *i* coordinates of the three spots of interest. The scaling factor for the top and bottom boundary computations is empirically determined to be 0.7. Likewise, the left boundary of the subsequent processing box is determined by subtracting the product of Δ_j and a scaling factor from the smallest *j* coordinates of the three spots of interest, and the right boundary of the subsequent processing box is determined by adding the product of Δ_j and the same scaling factor to the largest *j* coordinates of the three spots of interest. The scaling factor for the top and bottom boundary computations is empirically determined to be 0.2. A safeguard is incorporated in the function to disallow selection of a boundary that would be outside the image area. There are no instances of the spot of interest falling outside the predicted processing box when this process is executed on video clips of several different approaches. Results from one of the representative video clips are presented in figure 25. Note that the upper and lower limits of the y-axis in each of the plots corresponds to the respective scaling factor for that processing box dimension, and that none of the data exceeds the upper or lower limits of either plot.

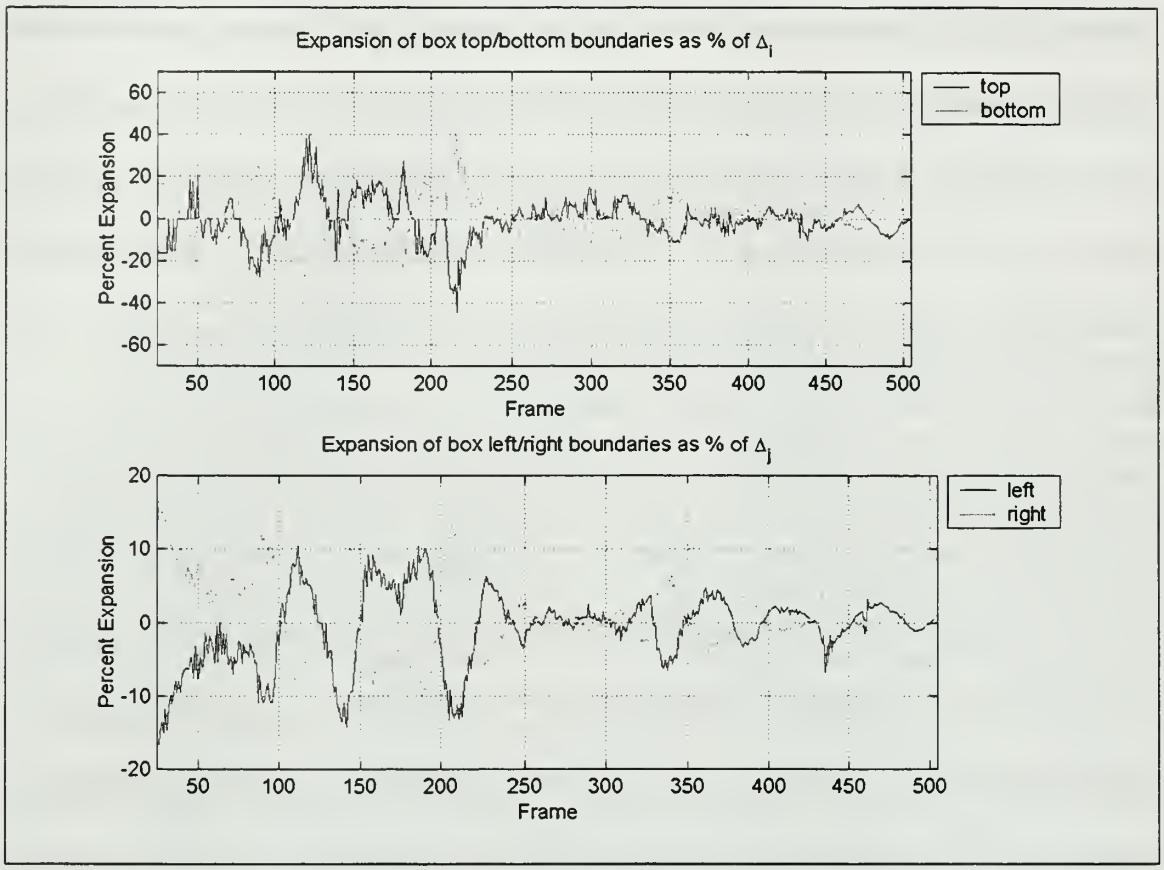

Figure 25. Search Box Expansion for Representative Video Clip

The trend of the processing box height and width, measured in pixels, through a full video clip of an entire approach is presented in figure 26.

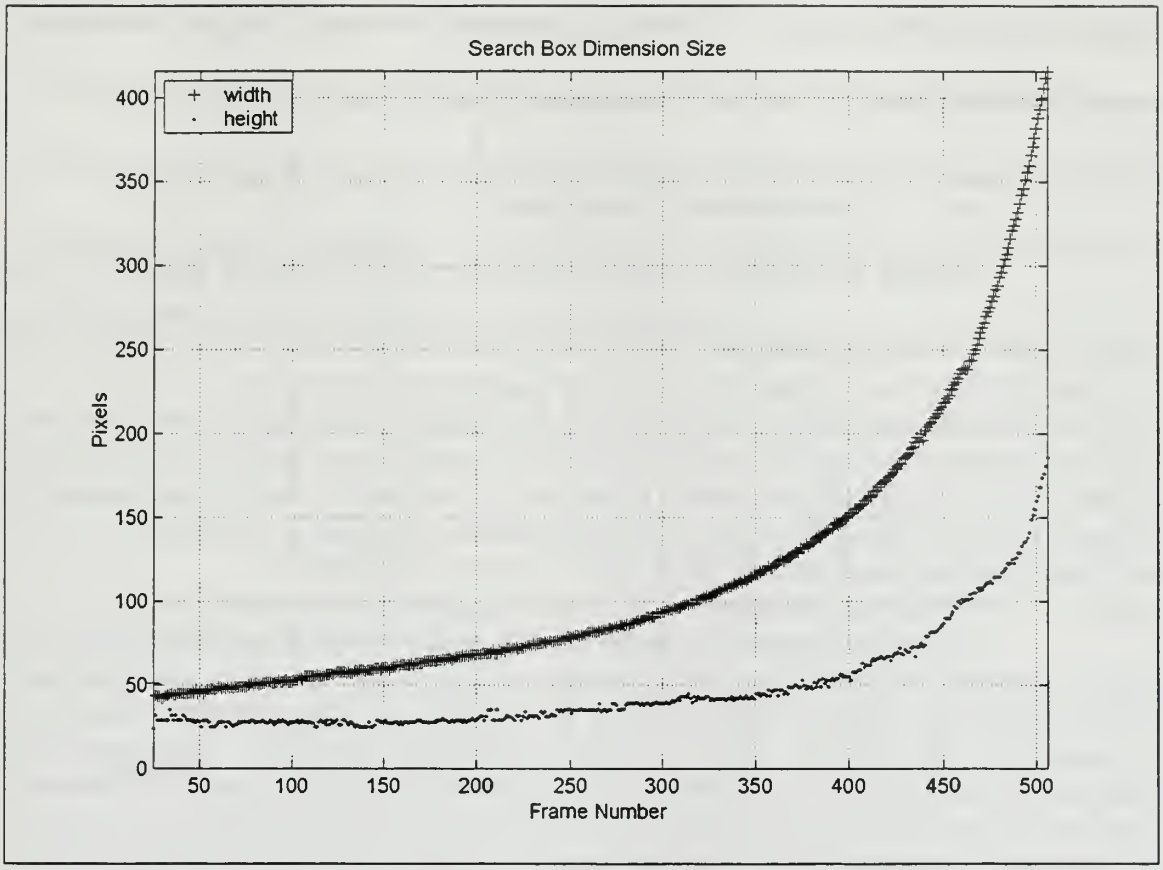

Figure 26. Search Box Size for Representative Video Clip

7. Image Plane Coordinates

Following variable and constant initialization and determination of the location and dimensions of the first processing box, an iterative loop is executed to: (1) load and normalize the next image in sequence, (2) locate the three spots of interest in the image and compute the image plane coordinates of their centers, (3) compute the dimensions and position of the processing box for the next image in the sequence. This loop is repeated for each image in the sequence to be processed. As the image plane coordinates are computed for the three spots of interest in each image, the most recent set of coordinates is appended to an array that includes all of the coordinate sets for the entire

sequence of processed images. Upon completion of the loop, this array contains a complete set of coordinates for the three spots in the entire image sequence.

a. Determination of spot center

Within the iterative loop of *controller.m*, the function *findcenter.m* is called for each image to locate the three spots of interest and compute their centers. This function initializes values for δ_i and δ_j as well as the upper and lower threshold limits and applies a bisection thresholding search to the area of the image defined by the processing box that is predicted as a result of the previous iteration of the loop.

An initial threshold is applied to the portion of the image contained within the processing box. All of the pixels in the processing box whose luminance exceed the threshold are assigned to a new array. If no pixels are found that exceed the threshold, the threshold is reduced to the value that bisects the lower half of the remaining range of allowable thresholds, and the new threshold is reapplied to the processing box. This process is repeated until the threshold is low enough that some pixels are found that exceed it. In practice, however, the initial threshold is chosen such that it is low enough to include pixels from all three spots of interest for the vast majority of the images.

The array that contains all of the pixels whose luminance exceeds the threshold is then sorted and the number of spots is determined as described previously. As individual spots are identified, the coordinates of their component pixels are removed from the array of all "bright" spots and saved to another distinct array, one for each spot as it is isolated. One additional mechanism is added in this implementation. Since it is desired to isolate exactly three spots, sorting and counting of pixels halts as soon as more

than three spots are identified at the current threshold. This prevents time from being wasted processing more pixels after it is determined that more than three spots are present at the current threshold level. The threshold is then raised using the bisection thresholding technique previously described, and the process is repeated until exactly three spots are extracted from within the confines of the processing box.

As implemented, dynamic threshold adjustments can be made until the difference between the upper threshold limit and the lower threshold limit is less than 0.005. Without such a limit, the bisection algorithm would iteratively halve the range of acceptable thresholds until the range equaled the computational precision of the host computer, which would result in an unacceptably long delay in individual image processing time. However, with such a limit in place, an additional mechanism is required to accommodate the case in which the bisection algorithm reaches the limit before exactly three spots are isolated from the portion of the image inside the processing box. This occurs in the latter portion of the landing approach, when the three spots appear their brightest, saturating the available range of pixel luminance.

By the time the dynamic threshold adjustments alone are inadequate to isolate the three spots, the sizes of the three spots (the number of pixels that comprised them) has increased significantly relative to what they are at the start of the approach when the aircraft is at greater range from the landing area. To accommodate the larger spot sizes observed in the later frames of the image sequence, the function includes a mechanism to dynamically increase the values of δi and δj when the range of allowable threshold values was reduced to 0.005 or lower. This effectively redefines the number of

adjacent pixels that comprise a spot as required as the spot size increases in the image as the aircraft approaches the landing area. Conditionally adjusting the working definition of a spot by increasing δi and δj yields an effective complement to the use of the luminance threshold to efficiently isolate the three spots of interest. The final threshold for each of the images in a representative video clip is presented in figure 27.

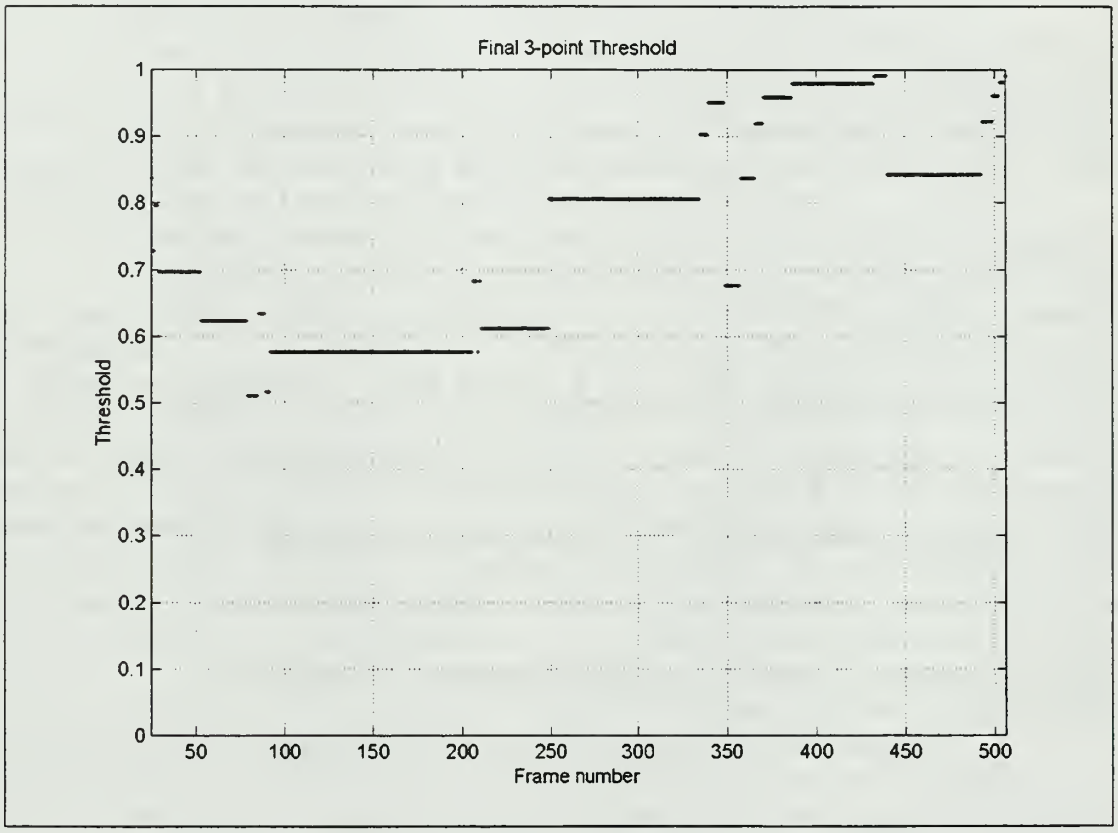

Figure 27. Final Threshold at Which Three Points Isolated

When exactly three spots have been isolated, the individual arrays of pixels for each spot were passed to the local function *weightedcenter.m* which computes the center of each spot using the weighting method described previously. These computed centers are then returned by *findcenter.m* to the function that originally calls it.

b. Processing box update

After the individual centers for each of the three spots of interest are computed and returned to *controller.m*, the local function *nextbox.m* updates the size and position of the processing box using the method previously described. The program then returns to the beginning of the iterative loop. With images in the sequence remaining to be processed, the loop executes; otherwise, the program performs the requisite sorting and transformation of the spot center coordinates in ij coordinates to image plane coordinates so the data is compatible with the numerical three point algorithm.

8. Data Transformation and Storage

a. Final sorting

The three point algorithm specifies the order in which the spot coordinates are passed to it. As they appeared in the image, the first point is the lower, rightmost vertex of the triangle defined by the three points. The third point is the leftmost vertex of the triangle, and the second point is the remaining vertex.

The local function *sortij123.m* accepts the array produced by the iterative loop that consists of the complete set of spot centers in ij coordinates for each frame in the sequence of processed images. Each row in the array represents one image frame and contains three coordinate pairs, one for the center of each spot in the corresponding image. However, for a given row, the coordinate pairs are in no specified order.

The *sortij123.m* function first locates the left most point, Point 3, by determining which of the three has the smallest column coordinate. Of the remaining two

points, the one with the greater row coordinate, and therefore the "lowest" in the image, is Point 1. The remaining point is Point 2.

Sorting is performed in this order to take advantage of the observation that Point 3 always appears as the leftmost point in the image and sometimes appears lower in the image than Point 1. Point 1 always appears lower than Point 2, but due to aircraft bank angle, it is not always lower than Point 3. By identifying Point 3 first based on its column position, Point 1 can then be conclusively identified from the remaining two points.

b. Conversion to Image Plane Coordinates

Once the spot center coordinates are placed in proper order, they are transformed from the ij coordinate system previously described to image plane coordinates. The local function *ij2uv.m* performs this straightforward transformation by algebraically shifting the origin of the ij coordinate system from the upper left corner of the image to the center of the image.

c. Data Storage

Once the entire sequence of images is processed, the frame number and image plane coordinates of the center of each of the three spots in each image frame are exported as a single matrix to a MATLAB data file.

9. Integration with the Three Point Algorithm

The image data file exported by the image processing algorithm is formatted in such a manner that it was fully compatible with a high-fidelity SIMULINK

implementation of Yakimenko and Kaminer's three point algorithm. By marrying the image processing MATLAB implementation with the SIMULINK implementation of the numerical three point algorithm, the feasibility of vision-based navigation can be demonstrated based on actual flight test data.

THIS PAGE INTENTIONALLY LEFT BLANK

V. FLIGHT TEST

A. UNMANNED AERIAL VEHICLE DESCRIPTION

1. Airframe Description

The flight vehicle used in this investigation was the US Army FOG-R Unmanned Aerial Vehicle (UAV). Built by BAI Aerosystems, Incorporated of Easton, Maryland, it was a high-winged monoplane with a pod-and-boom fuselage and a swept vertical fin and rudder and low mounted horizontal tail. One 10 hp 150 cc two-cylinder piston engine with a two-blade, fixed pitch wooden propeller was center-mounted on a pylon above the wing. The FOG-R had fixed tricycle landing gear. The FOG-R was radio controlled (RC) with a Futaba Pulse Width Modulation (PWM) transmitter similar to those used by sport RC modelers. Comprehensive descriptions of the airframe can be found in Froncillo, Komlosy, and Rivers [Refs 10, 11, 12]. The FOG-R is pictured in figure 28.

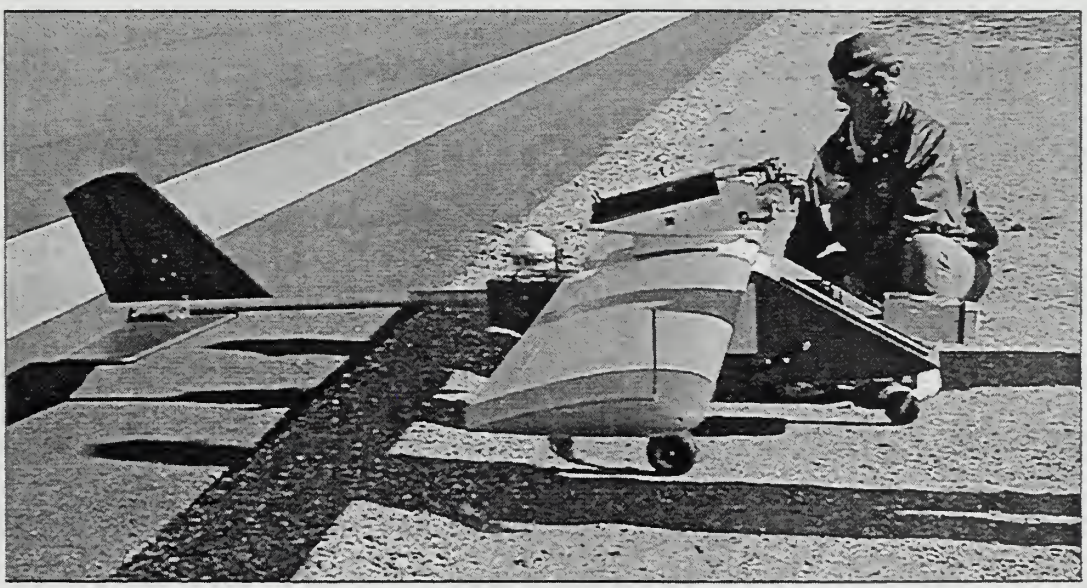

Figure 28. US Army FOG-R Unmanned Air Vehicle, Side View

Key airframe and performance parameters are presented in Table 1.

Parameter	Value
Takeoff Weight	90 lbs
Payload	25 lbs
Speed	90 ft/s
Ceiling	5000 ft
Wingspan	10.6 ft
Reference Wing Area	17.6 ft^2
Airfoil	NACA 2415
Maximum Lift/Drag Ratio	7

Table 1. US Army FOG-R UAV Airframe Characteristics

2. Sensor Description

a. Infrared Camera

The FOG-R UAV was equipped with an Infrared Components Corporation *MB IRES IMAGE CLEAR*™ Uncooled Microbolometer Module-based infrared (IR) video camera. The camera included a Boeing U3000A uncooled 8-12 μm sensor and the Microbolometer Module which produced National Television Standards Committee (NTSC) video signal and output it via a RS-232 interface. The focal length of the camera lens as installed in the FOG-R UAV was 25mm with a field of view of 40° x 30°. The pixel resolution of the camera video was 320 x 240. Computed pixel height as a function of range from the target is presented in figure 29.

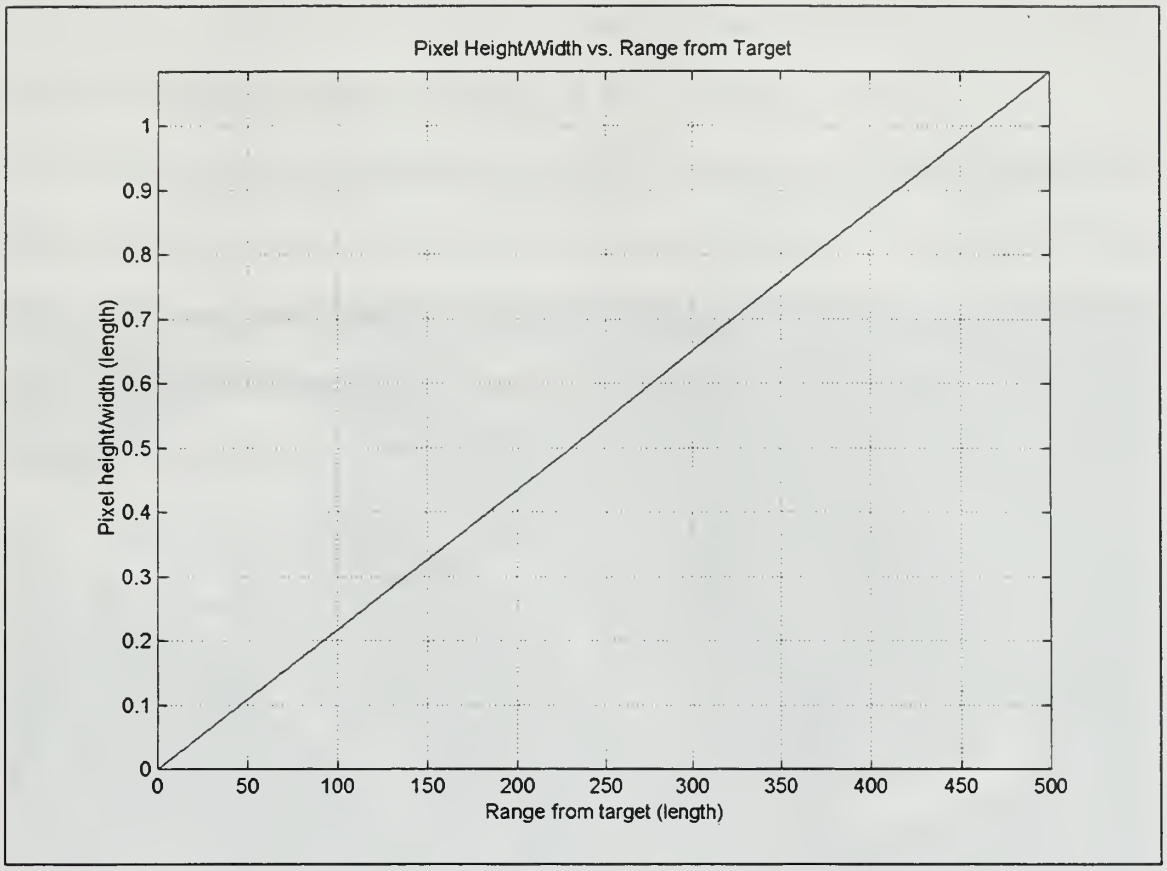

Figure 29. Pixel Height/Width vs. Range from Target

Additional information on the camera can be found in the *MB IRES IMAGE CLEAR*™ Operator's Manual [Ref. 13].

The camera was rigidly mounted in the nose of the aircraft, and the pointing angle was fixed in the x-z plane of the aircraft body axes, declined five degrees from the longitudinal axis of the aircraft. As a result of the fixed mounting in the aircraft, the aircraft heading and attitude alone determined the camera pointing angle. Because the focal length of the camera was fixed, the camera's field of view was fixed.

b. Video Tape Recorder

The video output of the IR camera was recorded in the FOG-R using a Sony Digital Video Walkman, model GV-D300. This digital video tape recorder (VTR) recorded the live NTSC format video signal from the IR camera in the Digital Video (DV) Standard format on a DV mini video magnetic cassette using a helical scan. In addition to the video image, the VTR also recorded the elapsed recording time of each video frame. Display of this elapsed timestamp on the image frame could be activated or deselected. Regardless of the chosen display option, the timestamp was available to the operator only visually during post-flight processing. No method of electronically extracting the timestamp data from the video tape for additional computation could be identified. The VTR as installed in the FOG-R weighed 21.5 ounces and consumed 6.2 W when recording.

c. Differential Global Positioning System

The FOG-R UAV was equipped with a Trimble AgGPS 132 Differential Global Positioning System (DGPS) system. The AgGPS 132 system consisted of a 12 C/A-code channel receiver, a combined GPS/DGPS receiver, and a ruggedized antenna cable. The receiver included ground beacon and satellite DGPS capability. The receiver produced Universal Time Code (UTC) stamped messages that included aircraft latitude, longitude, antenna height (altitude), GPS quality indication, number of satellites, Horizontal Dilution of Precision (HDOP), speed over ground, and magnetic variation. These messages were transmitted in ASCII format via 9600 Baud spread spectrum radio frequency (RF) data modems to a ground-based receiver which stored the received data

in text files. Additional information about the system may be found in the Trimble AgGPS 132 Operator's Manual [Ref. 14].

An illustration of sensor installation in the FOG-R UAV is provided in figures 30 and 31.

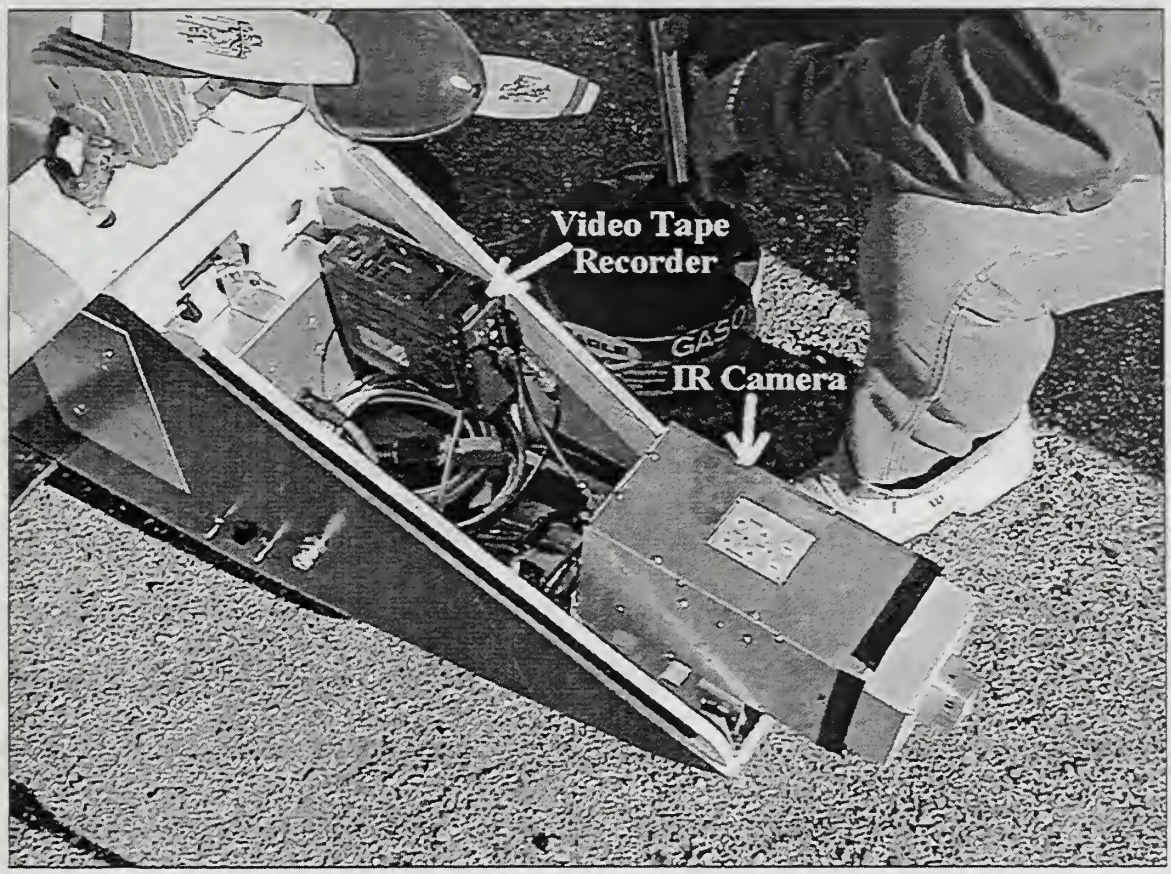

Figure 30. IR Camera and VTR Installation in FOG-R UAV

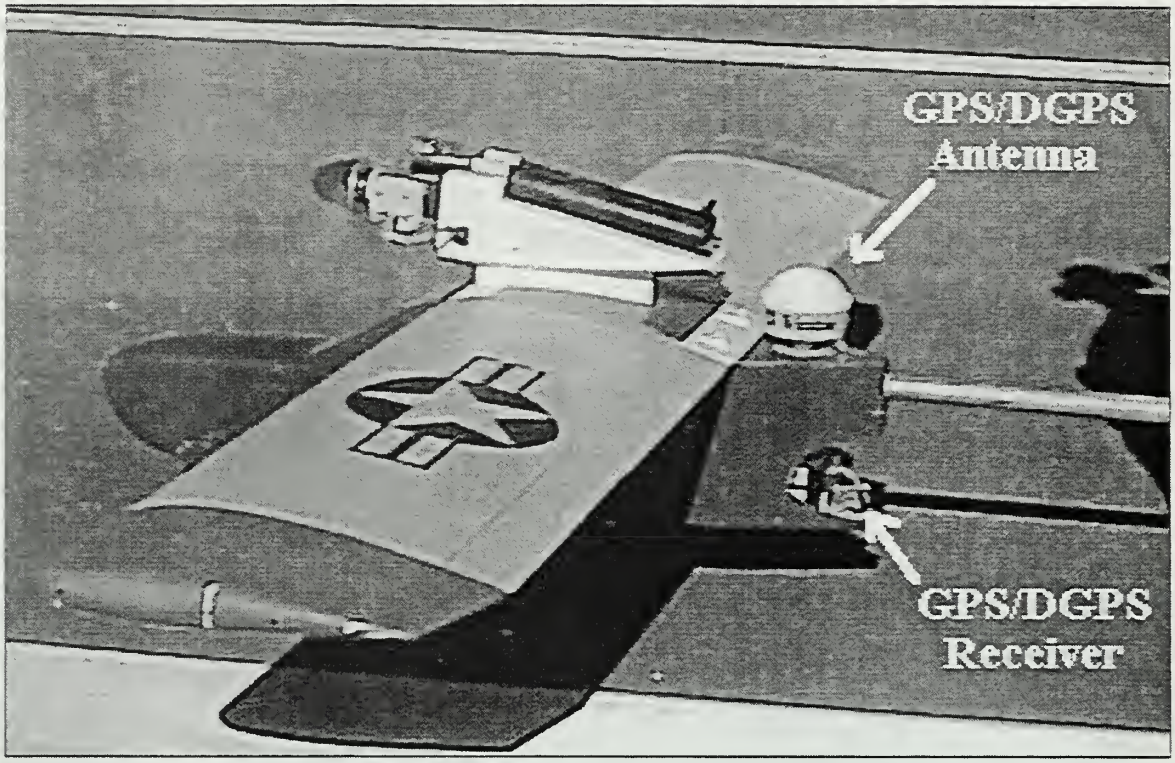

Figure 31. DGPS Installation in FOG-R UAV

B. FLIGHT PROFILES

As presented in reference 1, an observer's position in three dimensions can be unambiguously resolved if three visual reference points of know geometry and orientation are in the observer's field of view. This conclusion is fundamental to the determination of aircraft position and velocity estimates in the Local Tangent Plane (LTP) based on the observation of three reference spots on the ground. Recognizing this, live flight test profiles were constructed to represent a UAV landing on a specified landing spot with three distinct reference spots that appeared prominently when viewed with the IR video camera installed in the nose of the FOG-R. One flight test evolution was conducted, consisting of three sorties. Each sortie included a combination of several

low approaches and approaches flown all the way to landing touchdown. The majority of the approaches were flown using a left hand traffic pattern. The FOG-R was controlled manually via remote control at all times.

For the flight test evolution, three ordinary household charcoal grills were placed on either side of the runway in the proximity of the desired landing area, one grill on the left side of the runway and the other two on the right side. The positions of the grills relative to each other and relative to the ground were measured for post-flight data analysis, and charcoal fires were lit in each. The FOG-R was configured with the sensor suite described above, and video of each of the landing approaches was captured on video tape using the IR camera mounted in the nose of the aircraft and the digital VTR. Additionally, the time-stamped DGPS position and speed data of the FOG-R was recorded throughout its flights via a radio frequency modem downlink to a laptop computer. A radio frequency downlink of the video imagery was attempted, but frequent dropouts of the signal prevented it from being maintained consistently. The primary goal of the first flight test evolution was to assess the operation of the IR video camera in the flight environment and record video imagery of the landing approaches on the digital VTR installed in the aircraft. There was no intention to perform real-time processing of the video images during the first flight test period, but it was hoped that in later flight tests that this could be investigated as well. Unfortunately, due to project constraints, it was not possible conduct any additional flight tests during the timeframe of this investigation.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. RESULTS

A. POST FLIGHT PROCESSING

Using the data collected during the flight test evolution, the implementation of the algorithms presented in Chapter IV was refined to develop the final image processing program. The program processed video clips from several landing approaches during the months following the flight test evolution, and representative results are presented here.

B. SPOT LOCATION

A composite plot of the row- and column-wise positions of the centers of each of the three spots of interest in each frame is provided in figure 32. The spot centers are plotted as dots with different shades, and the borders of the processing box are plotted as solid lines. The left panel displays the row coordinate for each the three spot centers versus frame number. In this image sequence the first frame number processed is frame 25, and the final frame in the sequence is frame 506. The right panel presents the column coordinate for each the three spot centers versus frame number for the same image sequence.

Two observations of great significance are made of this plot. First, for every frame, there is a complete set of coordinates for each of the three spots. That is, there are no drop outs in the entire sequence. Secondly, for every frame, the three spots are located entirely within the horizontal and vertical boundaries of the processing box, thereby validating the bisection thresholding and processing box size and position determination algorithms.

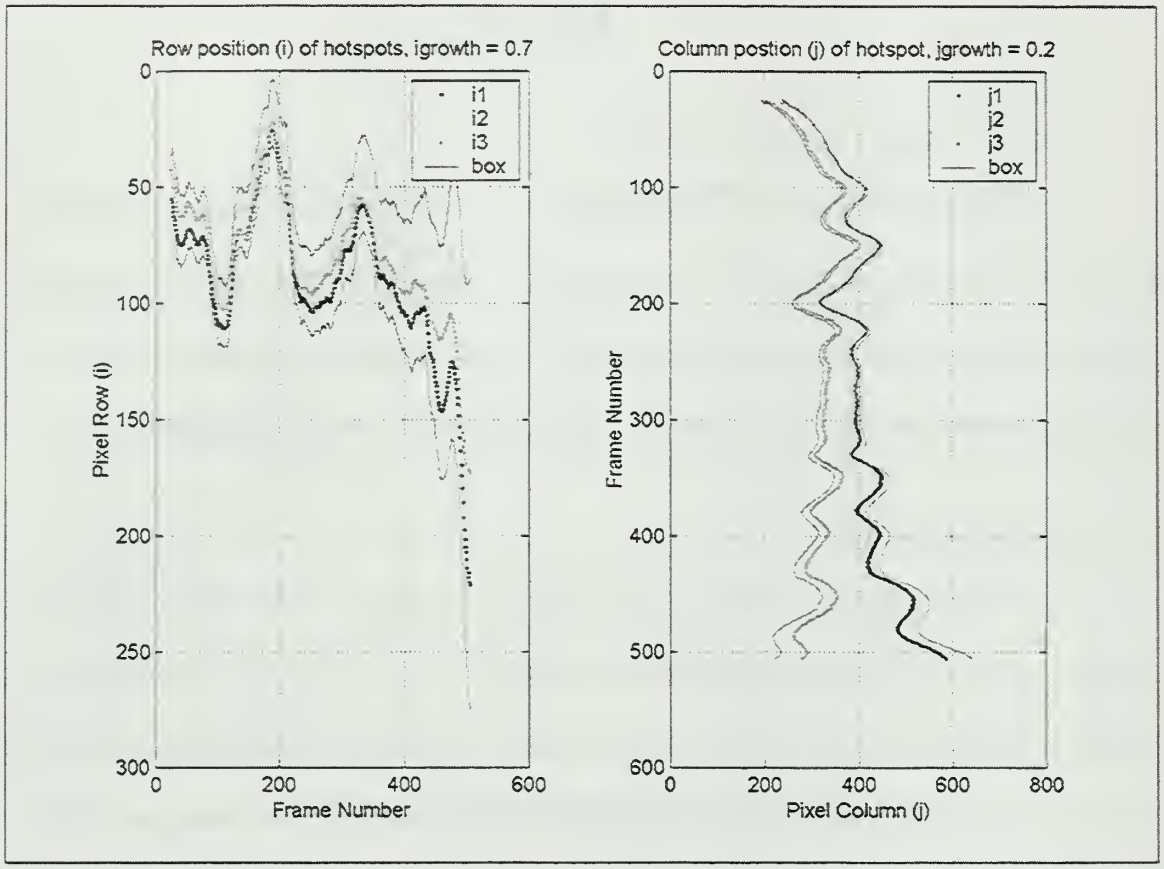

Figure 32. Spot Position and Processing Box Borders For Representative Approach

C. COMPARISON WITH DGPS

With a complete set of image coordinates from a processed video sequence, the aircraft position in the local tangent plane can be estimated using the three point algorithm. For analysis of the vision-based navigation algorithms already discussed, the aircraft position estimates as determined from the vision-based algorithms were compared with the aircraft position estimates determined from the Differential Global Positioning System data for the same approach. Unfortunately, the sensor package installed in the UAV did not possess a means to synchronize each sensor to a common clock, so it was not possible to timestamp the data collected from each sensor using a

common time reference. This significantly limited error analysis by preventing a direct comparison of the sensor data based on time of collection. Instead, the two sets of position estimates are plotted in figure 33 versus position vice time with the axes scaled to maintain a 1:1 proportion. Superimposed on the data plots are graphical representations of the runway edges and the landing area, represented by the measured hotspot locations, as indicated in the figure legend. The upper panel represents the lateral position (north-south vs. east-west) in the local tangent plane, whereas the lower panel represents altitude (negative z-direction) vs. the y (east-west) axis.

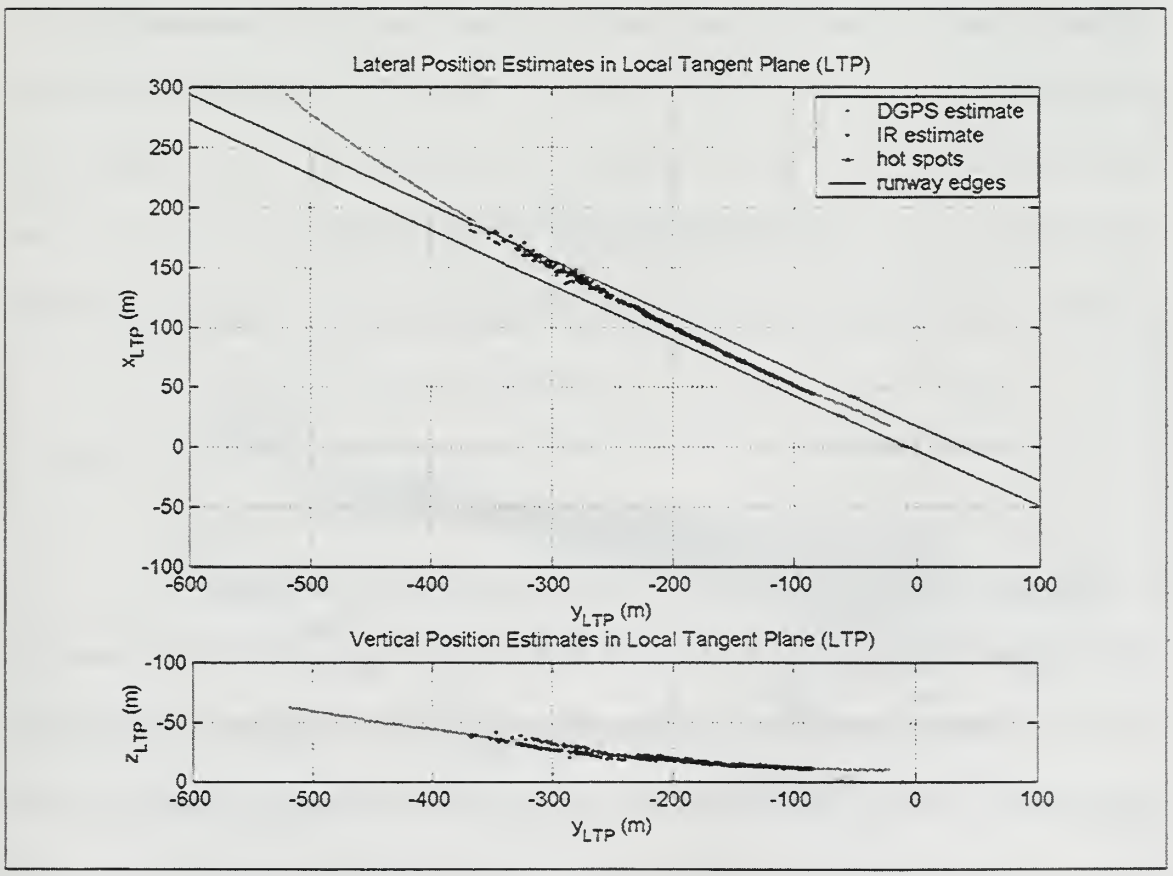

Figure 33. Comparison of IR and DGPS Position in Local Tangent Plane (2-D)

Despite the lack of a synchronous timestamp, there is clearly a close correlation between the DGPS position estimates and the vision-based position estimates in lateral and vertical position. There appears to be a slightly greater difference between the two estimates at greater ranges from the landing area. However, as the range from the vehicle to the landing area decreases, the correlation between the DGPS and vision-based position estimates appears to decrease. Also of note from figure 32 is that the greatest rates of horizontal and vertical change of the spots in the image plane, which correspond to higher aircraft yaw and pitch rates respectively, occur during the first half of the approach. Figure 34 presents a quasi three-dimensional perspective of the same data set.

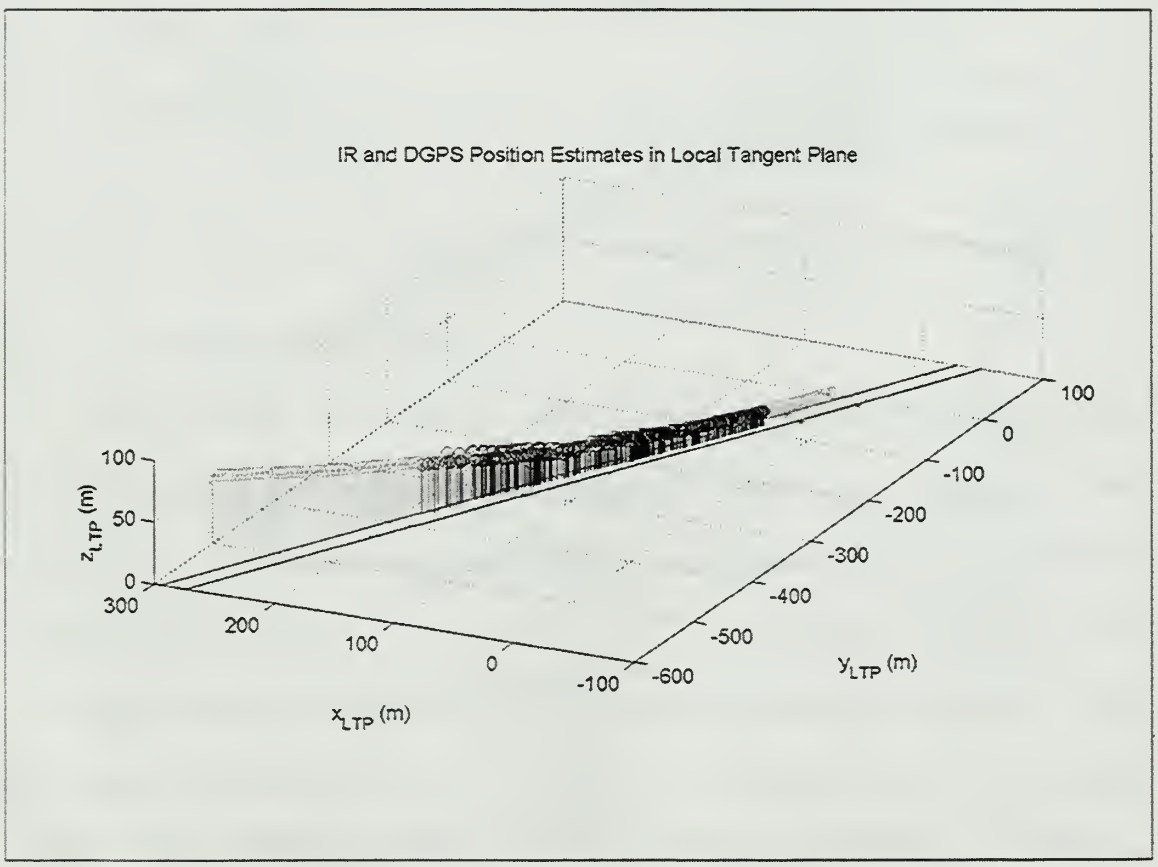

Figure 34. Comparison of IR and DGPS Position in Local Tangent Plane (3-D)

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

1. General

The primary objective of this thesis was accomplished; the feasibility of using vision-based sensors for aircraft navigation in support of autonomous landing has been demonstrated. A live flight test was conducted to capture video imagery of a UAV landing taken using an infrared video camera mounted in the aircraft's nose. Algorithms were developed to extract the relevant visual features that defined the landing area from the video and use the extracted data to compute estimates of the UAV's position in the local tangent plane. The aircraft position estimates based on the vision sensor were shown, within the limits of the test, to correlate with those produced by the Differential Global Positioning System, which were considered the truth reference for this evaluation.

2. Specific

a. Image Processing Algorithms

The image processing algorithms were developed in the spirit of demonstrating the feasibility of vision-based navigation using actual flight video imagery collected by an aircraft. Generality was maintained in the algorithms where possible but not at the expense of demonstrating the results of the flight test evolution. It is hoped that the reader will appreciate the feasibility of vision-based navigation and see fit to utilize, modify or expand the algorithms presented herein to satisfy his own application.

Two different processing methods were combined to address the respective challenges of processing the first image of a sequence with little *a priori* knowledge and then processing the subsequent images utilizing the knowledge gained from the previous image. By approaching the problem in this manner, the time required to process each image was minimized within the context of this investigation.

b. Real-time Processing

All of the video imagery was processed offline, after the flight evaluation. With an average time required to process the first image in the sequence approximately 35 seconds, and the average time to process the subsequent images approximately 0.5 seconds, real-time data processing is not achieved by the algorithms presented here.

c. Flight Test

A tremendous amount of data and findings were gleaned from the flight test evolution. With the FOG-R sensor package limited to the IR camera and the DGPS, navigation analysis was effectively limited to position data. However, without a common time signal in the sensor package from which all sensor data could receive a common, synchronized timestamp, error analysis of the position estimates from each of the sensors was severely restricted.

This flight test utilized a stationary landing platform, namely, a defined area on a land-based runway. While this is a logical starting point for an investigation of this topic, the maritime application of vision-based navigation is for providing relative position and velocity information between the aircraft and a moving landing platform.

B. RECOMMENDATIONS

1. Future Investigations

This investigation has merely scratched the surface of the realm of vision-based navigation for autonomous landing. Numerous other investigations should be made to broaden the understanding of vision-based navigation. A more detailed examination of source of error in vision-based position and velocity estimates should be made. The influence of pitch and yaw rates as well as range from the landing area should be investigated. Image processing algorithms that more robustly address out-of-frame events should be developed. The influence of out-of-frame events on the navigation solution should be investigated. The blending of all passive navigation sensors to achieve an optimum estimate of the vehicle's navigation state vector should be pursued.

2. Flight Test

Since only one flight test evolution was performed, it is strongly recommended to continue flight testing to refine current algorithms. The quantity and quality of data obtained and the findings that were from the initial flight test are a strong endorsement for the value of flight test. Fight test is a must to fully characterize system performance in the investigations proposed in the previous chapter. Flight test utilizing a moving landing platform is strongly recommended as appropriate for the UAV sensor system's maturity.

3. UAV Enhancements

Two significant enhancements to the FOG-R UAV should be made as soon as possible to maximize the gain from future flight test. An Inertial Navigation System should be installed to measure thrust accelerations and angular velocities in support of future investigations. Equally important, a common, synchronized timestamp of all sensor data is essential to perform any meaningful error analysis in future investigations.

VIII. REAL-TIME CONTROLLER HARDWARE INTERFACE

A. INTRODUCTION

The secondary purpose of this thesis is to present a modification of the Naval Postgraduate School's Rapid Flight Test Prototyping System (RFTPS) to support NPS' development and optimization of the guidance system for the U.S. Air Force's Affordable Guided Airdrop System (AGAS).

The Air Force has identified a critical need to improve the accuracy of materiel airdrop. Significant emphasis has been placed on the development of large-scale parafoil systems. These systems provide the requisite accuracy for precision airdrop, but their cost per pound is prohibitive. Alternative, low-cost systems are currently under investigation by the U.S. Air Force. [Ref. 15]

Currently under design and development is the AGAS, which incorporates a low-cost guidance, navigation, and control (GNC) system into fielded cargo systems. The design goal of the AGAS is to provide a GNC system that can be placed in-line with existing fielded cargo parachute systems and standard delivery containers. The current design concept includes a navigation system and guidance computer that would be secured to the existing container delivery system. Pneumatic Muscle Actuators (PMAs) would be attached to each of four parachute risers and to the container and would effect control by extending or contracting on command from the guidance computer. [Ref. 15]

The GNC system is a collective effort of students, scientists, and engineers at the Naval Postgraduate School and Vertigo, Incorporated; the NPS contribution is the

optimization and testing of the parachute system [Ref. 16]. The RFTPS is crucial to this effort.

The NPS RFTPS has been used successfully in conjunction with the FOG-R UAV to support research and development of many student projects and theses. However, the AC100/C30 real-time controller utilized by the RFTPS in previous projects has become obsolete and been replaced by an updated device, designated AC-104. Both real-time controllers are produced by WindRiver® (formerly, Integrated Systems, Incorporated), and are used in conjunction with that company's MATRIX-X® modeling, simulation, and rapid prototyping software package. The RFTPS with the AC-104 real-time controller has not yet been used to support live flight test events. The updated real-time controller incorporates two new input/output (I/O) modules that must be characterized before it can be successfully used in support of AGAS development. The body of this chapter is primarily concerned with characterizing the I/O modules of the updated real-time controller.

B. SUMMARY OF PREVIOUS WORK

As discussed, the RFTPS has been successfully used in conjunction with several previous efforts. The motivation and design considerations for developing the RFTPS as well as a detailed system description are well documented by Hallberg, Kaminer, and Pascoal [Ref. 17]. Dellicker assesses the feasibility of the AGAS concept and presents initial flight test data as well simulation results that demonstrate the strong potential of the AGAS to meet the requirements of low-cost precision airdrop [Ref. 15]. Williams expands Dellicker's work by moving Dellicker's model from a MATLAB/SIMULINK®

environment to a MATRIX-X® environment and refining it in anticipation of hardware-in-the-loop analysis and flight test utilizing the AC-104 real-time controller [Ref. 16].

C. REAL-TIME CONTROLLER

1. Rapid Flight Test Prototype System Ground Station

The RFTPS ground station is responsible for flight control and data collection, and consists of a host computer/real-time controller, a communications box, and two Futaba RC controllers.

The heart of the ground station is the real-time controller. The AC-104 hardware controller currently used in the RFTPS replaces the AC100/C30 system used in the original implementation of the RFTPS [Ref. 17]. A Windows NT based personal computer (PC) serves as the host computer and is networked with the AC-104 via Ethernet. The host computer is used to perform all functions necessary to generate executable code which is downloaded from the host PC and run on the AC-104.

The communications box contains all equipment necessary for communication of control commands and flight data between the airborne vehicle and the ground station. This includes two RF modems, a GPS receiver, and a Futaba pulse wave modulation (PWM) receiver identical to what is installed in the airborne vehicle. The airborne vehicle is controlled using two Futaba RC controllers. One controller, referred to as the "slave", is modified to accept inputs from the digital to analog module installed in the real-time controller via a 9-pin, RS-232 connector. The slave converts the voltages it receives as analog input from the real-time controller to properly formatted PWM signals.

The slave then forwards the PWM signals to standard Futaba controller, referred to as the "master", from which the commands are transmitted via radio frequency to the airborne vehicle. The slave controller is connected to the master via a production Futaba hard line data link cable. A safety pilot may intervene and assume manual control of the airborne vehicle by releasing a spring-loaded trainer switch on the master controller. The trainer switch must be actively held open to for the master to accept and transmit inputs from the slave. Releasing the trainer switch causes the master controller to disregard inputs from the slave and accept manual inputs from the safety pilot via the controls mounted on the master. [Ref. 3]

2. MATRIX-X® Software Family

Installed on the host PC, the MATRIX-X® software family includes several individual, yet related, applications. Xmath is the computational element of the package, and SystemBuild provides modeling and simulation functionality by using predefined and user-defined functional blocks to model system elements. AutoCode is an application that generates C++ source code from a SystemBuild model. An animation builder enables the user to build a Graphical User Interface (GUI) that allows real-time inputs and monitoring of system parameters when the controller is running. The hardware connection editor is used to designate connections between the I/O ports on the front of the AC-104 and data paths within the code running on the controller. The RealSim environment allows models developed in SystemBuild to be run in real-time, connecting to real hardware for real-time simulation, rapid prototyping, and hardware-in-

the-loop modeling. The RealSim environment is managed using the GUI depicted in figure 35. [Ref. 18]

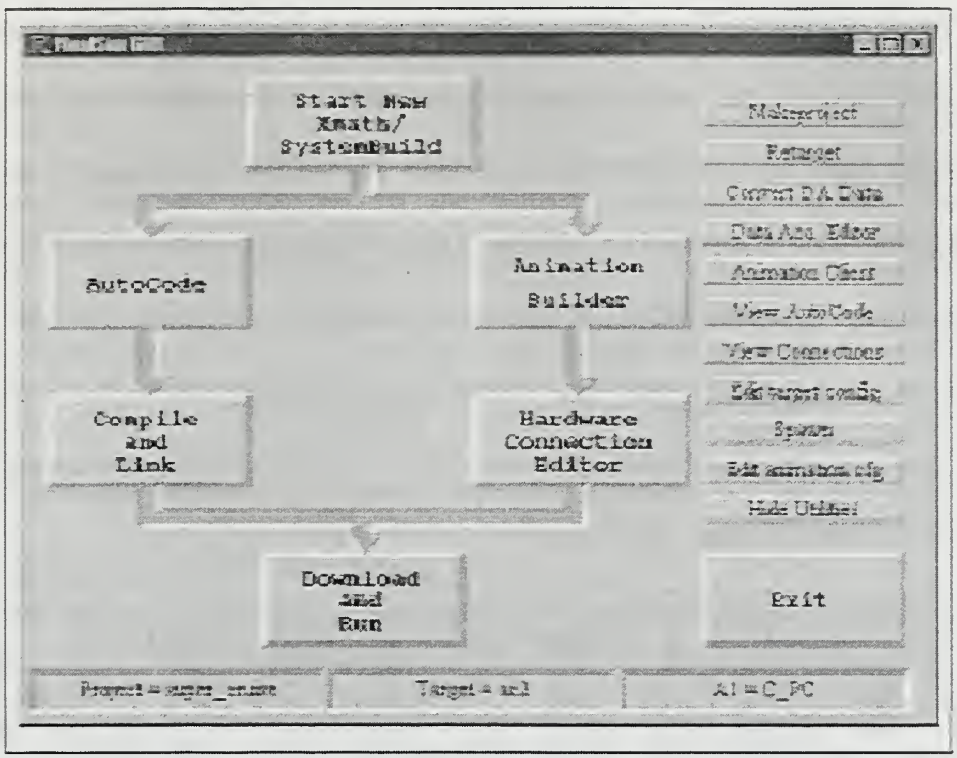

Figure 35. RealSim Graphical User Interface [Ref. 18]

As can be seen from figure 35, the RealSim GUI provides a flow chart approach to the process of developing an executable file to be run on the AC-104, also referred to as the target controller. Once the left and right paths of the flow chart are completed, the RealSim software on the host PC generates an executable code which is downloaded to the target controller via file transfer protocol (FTP). Detailed instructions for building a new model are presented in section 3.6 of reference 18. Detailed instructions for building a GUI for a new model using the animation builder are presented in section 4.3 of reference 18, and the remaining step reflected in the RealSim GUI are presented in detail in chapter 5 of reference 18.

3. AC-104 System Description

The RealSim AC-104 real-time hardware controller is based on a small, 8" x 5.75", highly integrated PC motherboard that includes a PC/104 expansion connector. It uses PC/104 I/O boards and optional Industry Pack (IP) modules mounted on the Flex/104 IP carrier for the PC/104 bus [Ref. 19]. The AC-104 configuration used in the RFTPS ground station included an AIM16/12 (AIM1612) 16 channel analog input board installed in port P1, an IP-68332 is a general purpose 68332 micro-controller module installed in port P3, an IP-Serial Port module installed in port P6, and a Ruby-MM 8 channel analog output board installed in port P8. The AC-104 front panel arrangement is presented in figure 36. [Ref. 18]

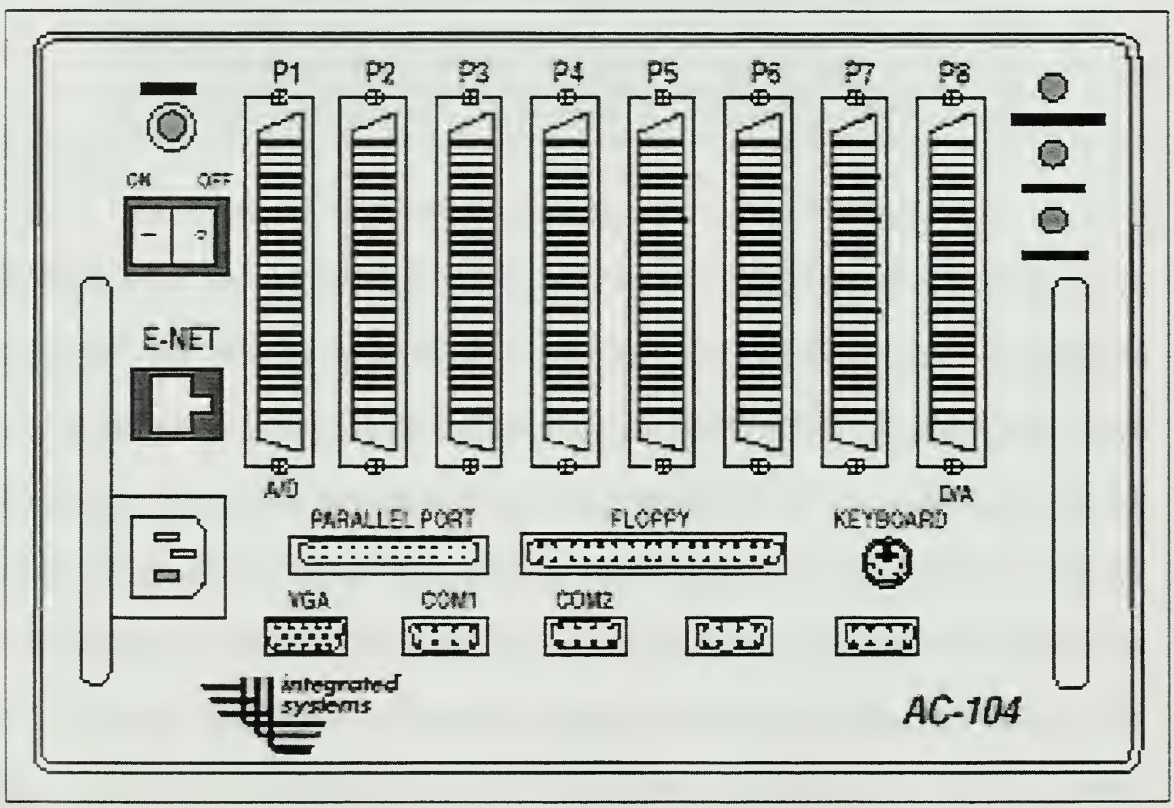

Figure 36. AC-104 Front Panel Arrangement

a. Ruby 8 Digital to Analog Converter Module

The Ruby-MM digital-to-analog converter (DAC) module in the AC-104 replaced the IP-DAC module installed in the C-30 real-time controller. In order utilize the existing command channels of the Futaba controllers and the established configuration of the communications box wiring, it was necessary to determine the mapping of the Ruby-MM DAC front panel connector pin arrangement to that of the IP-DAC as installed in the C-30 real-time controller. The Ruby-MM DAC front panel connector pin arrangement is presented as table 6-6 in section 6.3.5 of reference 19, and relevant portions are repeated here in table 2.

Futaba Slave	**SCSI Cable**	**Ruby DAC I/O** (AC-104 Front Panel)	**Function**	**IP-DAC** (C-30 Front Panel)	**C-30 to Comm Box Cable**
DB-9[1]	-	50-pin[3]	-	50-pin[3]	-
Pin #	Wire #[2]	Pin #	-	Pin #	Wire #[2]
1	2	26	DAC Ch 1	2	3
2	4	27	DAC Ch 2	8	15
3	6	28	DAC Ch 3	14	27
4	8	29	DAC Ch 4	20	39
5	Disconnected	-	-	-	-
6	Disconnected	-	-	-	-
7	Disconnected	30	DAC Ch 5	26	2
8	Disconnected	31	DAC Ch 6	32	14
9	1,3,5,7	1,2,3,4	Analog GND	1,7,13,19,25,31	1,13,25,37,49,12

1 – Pins 5,6,7,8 not connected to DB-9 connector installed in Futaba Slave Unit
2 – SCSI 50-pin ribbon cable-numbering convention used
3 – Centronix pin numbering convention used

Table 2. Pinout Map for Conversion from C-30 IP-DAC to AC-104 Ruby DAC

The legacy cable from the C-30 based RFTPS that was used to connect the 50-pin connector from the IP-DAC to the 9-pin connector on the Futaba slave was analyzed using a multimeter, and the findings are presented in table 2. The functions of

the active pins in the IP-DAC cable were determined from an archived copy of the IP-DAC front panel connector pin arrangement to be DAC channels 1 through 6. Inspection of the Futaba slave revealed that only pins 1 through 4 and 9 are connected on the 9-pin serial connector installed on the controller. The corresponding pins for the six DAC channels on the front panel connector for the Ruby-MM DAC were identified, and new cable was constructed to connect the correct DAC channels from the AC-104 Ruby-MM DAC connector to the Futaba slave. The associated pin and wire numbers are listed in the three leftmost columns of table 2. Note that only DAC channels 1 through 4 are connected with the new cable due to the Futaba slave connector configuration.

The new cable was verified to be correct by connecting the AC-104 to the RFTPS communications box and Futaba controllers and running a know executable on the real-time controller. Because the original executable was developed for the IP-DAC of the C-30, the executable was modified via the Hardware Connection Editor (HCE) by designating the Ruby-MM DAC as the output device vice the IP-DAC. The appropriate offset and scaling factor were entered for a voltage range of 0 to 5 volts in accordance with table 6-3 of section 6.3.1 of reference 19. Additionally, the J4 hardware jumpers were set appropriately for the 0 to 5 volt range in accordance with table 6-5 of section 6.3.3. When the updated executable was run, the expected results were observed.

b. AIM Analog to Digital Converter Module

The AIM analog-to-digital converter (ADC) input module was also investigated in anticipation of receiving analog voltages from four pressure transducers

installed in the AGAS, one for each PMA. A simple model consisting of a single unity gain block was constructed using SystemBuild. The AIM ADC was designated as the input device via the HCE, and a simple GUI was built to display the numerical values of the input voltages. A cable was built to connect the AIM ADC to a voltage reducer that interfaces with the pressure transducers on the AGAS. Table 6-21 in section 6.5.6 of reference 19 was consulted to determine the correct pin arrangement for the cable. The board was operated using single-ended channels and the default bipolar voltage range of ±10 volts. With the executable running on the AC-104, all four of the PMAs were inflated and deflated. The voltages representing PMA pressures were displayed on the controller GUI and corresponded well with expected values.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Yakimenko, O., Kaminer, I., and Lentz, W., "A Three Point Algorithm for Attitude and Range Determination using Vision," paper presented at the American Control Conference, Chicago, Illinois, 28 June 2000.

2. Kaminer, I.I., *AA4276: Avionics System Design*, Lecture Notes, Naval Postgraduate School, Monterey, CA, January 2000.

3. Watson, Mark T., *Vision Guidance Controller for an Unmanned Aerial Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1998.

4. Schmidt, L., *Introduction to Aircraft Flight Dynamics*, pp. 2-3, American Institute of Aerodynamics and Astrodynamics, 1998.

5. Haralick, R.M., Lee, C., Ottenberg, K., Nölle, M., "Analysis and Solutions of the Three Point Perspective Pose Estimation Problem," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1991, pp. 592-598.

6. Hallert, B., *Photogrammetry*, McGraw-Hill Book Company, Inc., 1960.

7. Eastman Kodak Company, "Digital Learning Center." [http://www.kodak.com/US/en/digital/dlc/book4/chapter1/index.shtml]. July 2000.

8. The Math Works, Inc., *MATLAB Image Processing Toolbox User's Guide, Version 2*, 1998.

9. Kuhn, K., "Conventional Analog Television." [http://www.ee.washington.edu/conselec/CE/kuhn/ntsc/95x4.htm]. May 1996.

10. Froncillo, S.J., *Design of Digital Control Algorithms for Unmanned Air Vehicles*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1998.

11. Komlosy, J.A., *Applications of Rapid Prototyping to the Design and Testing of UAV Flight Control Systems*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1998.

12. Rivers, T.C., *Design and Integration of a Flight Management System for the Unmanned Air Vehicle FOG-R*, Engineer's Thesis, Naval Postgraduate School, Monterey, California, December 1998.

13. Infrared Components Corporation, *MB IRES IMAGE CLEAR™ Operator's Manual*, Utica, New York.

14. Trimble GPS, *AgGPS 124/132 Operator's Manual.*

15. Dellicker, S.H., *Low Cost Parachute Guidance, Navigation, and Control*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1999.

16. Williams, T.A., *Optimal Parachute Guidance, Navigation, and Control for the Affordable Guided Airdrop System (AGAS)* , Master's Thesis, Naval Postgraduate School, Monterey, California, June 2000.

17. Hallberg, E., Kaminer, I., and Pascoal, A., "Development of a Flight Test System for Unmanned Air Vehicles," IEEE Control Systems, v. 19, pp. 55-65, February 1999.

18. Integrated Systems, Incorporated, *RealSim User's Guide*, Sunnyvale, California, February 1999.

19. Integrated Systems, Incorporated, *RealSim PC Controller System Reference*, Sunnyvale, California, February 1999.

APPENDIX A. DOCUMENTED MATLAB CODE

1. CONTROLLER.M

```
%  LCDR Paul A. Ghyzel, USN
%  September 2000
%  Thesis:  "Vision-Based Navigation for Unmanned Air Vehicles"
%
%  This program is intended to determine the X-Y coordinates of the centroids
%   of the three "hot spots" used for visual navigation of the FOG-R UAV.
%
%  SUBROUTINE HEIRARCHY
% 1a. averagebox.m
%     a. imnorm.m
%     b. showbrightest.m
%     c. nextbox.m
% 1b. filterbox.m
%     a. imnorm.m
%     b. imfilter.m
%     c. findxy.m
%        i. finddiff.m
%     d. plotstem.m
%     e. nextbox.m
% 1c. definebox.m
%     a. show_peaks.m
%        i. imnorm.m
%  2. imnorm.m
%  3. findcenter.m
%     a. weightedcenter.m
%  4. plotcentroid.m
%  5. nextbox.m
%  6. sortij123.m
%  7. ij2uv.m
%
%  Analysis Plots:
%   8. plotcenter.m
%   9. plotthreshold.m
%  10. plotboxheightwidth.m
%  11. plotimagetime.m
%  12. plotthreshtime.m
%
%  Additionally, the images to be processed must be:
%  1. located in the same folder as these program files;
%  2. 480 x 640
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
clear
delete controller.dat
diary controller.dat
tic
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  PROGRAM CONSTANTS
%
filename = 'picture';
filetype = '.bmp';    % '.bmp', '.jpg', '.tif'
igrowth = .7;         % percent of axis norm by which search box grows;
                      % "short" set up -> 0.7, "long" -> 0.35
jgrowth = .2;         % default -> 0.2
vtol = 6;             % vertical pixel tolerance:  # of pixels considered in
same spot
htol = 4;             % horizontal pixel tolerance, these values also hard-coded
                      % in findcenter.m
threshold = .9;       % initial threshold

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  INITIALIZATION
%
first_img   = 25;     % number of first image to process (25)
last_img    = 506;    % number of last image to process  (506)
%
no_imgs     = last_img - first_img + 1;    % total number of images to process
img_no      = first_img;
center = [];
box = [];
centroidtime = [];
img_time = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  READ IMAGE FROM FILE AND NORMALIZE
%
I = imread([filename, num2str(img_no), filetype]);
I = imnorm(I);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  DEFINE FIRST PROCESSING BOX
%
newbox = averagebox(I, filename, img_no, filetype);     % no a priori knowledge
%newbox = filterbox(I, filename, img_no, filetype, vtol, htol);
   % auto. define 1st box
%newbox = definebox(I, filename, img_no, filetype);   % manually define 1st box

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  PROCESS SEQUENCE OF INDIVIDUAL IMAGES
%
start_process = toc;
while img_no < (first_img + no_imgs)
   clc
   fprintf(['\n Processing image:  ',num2str(img_no-first_img+1),'  of
            ',num2str(no_imgs)])
   start_img = toc;
      I = imread([filename, num2str(img_no), filetype]); % read image from file
      I = imnorm(I);         % converts intensity values to scale from 0 to 1
      box = [box; newbox];   % captures search box parameters for current image
     [brightpoints,peakcenter,threshold,vtol,htol,centroidtime] =
         findcenter(threshold,I,newbox,vtol,htol,centroidtime);
      center = [ center; img_no peakcenter(1,:) peakcenter(2,:) peakcenter(3,:)
                threshold];
   stop_img = toc;
   img_time = [img_time; stop_img-start_img];
   % plotcentroid(filename, filetype, img_no, brightpoints, newbox, threshold,
```

```
                peakcenter)
   img_no = img_no + 1;
   newbox = nextbox(peakcenter,iggrowth,jgrowth);  % resize search box
end %while
elapsed_time = toc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  DATA TRANSFORMATION
%
centerij = sortij123(center);   % sorts points into 1-2-3 order for 3 pt alg.
centeruv = [centerij(:,1) ij2uv(centerij(:,2:3)) ij2uv(centerij(:,4:5))
            ij2uv(centerij(:,6:7))];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  ANALYSIS
%
fprintf(['\n\n Number of images processed:    ', num2str(no_imgs)])
fprintf(['\n Mean time per image:    ', num2str(mean(img_time)),' seconds'])
fprintf(['\n Total elapsed time:    ', num2str(fix(elapsed_time/60)),' minutes
        '])
fprintf([num2str(((elapsed_time/60)-fix(elapsed_time/60))*60),' seconds\n'])

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  SUMMARY PLOTS
%  Note:  Comment these out to save time if performing multiple runs
%
%plotcenter(centerij,igrowth,jgrowth,box)
%plotboxheightwidth(box,no_imgs,first_img)
%plotthreshold(centerij, first_img, no_imgs)
%plotimagetime(first_img, no_imgs, img_time)
%plotthreshtime(first_img, center, no_imgs, img_time)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  SAVE DATA TO FILES
%  Note:  Comment these out to protect good previously computed data
%
%delete centerij.mat
%save centerij centerij
%delete centeruv.mat
%save centeruv centeruv

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
diary off
```

2. AVERAGEBOX.M

```
function newbox = averagebox(I, filename, img_no, filetype)

clc
start = toc;
fprintf('\n Searching for hotspots. \n')
points=5;
spot=7;
xpic=640;
ypic=480;
smooth=3;               %  half-width of siding window
smoothmin=1+smooth;     %  4
smoothmax=xpic-smooth;  %  637
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
for i = ypic:-1:1                  %   index from bottom to top of image (480->1)
   row=I(i,:);                     %   individual row of pixel intensity values
   av=mean(row);
   row=max(row-av,0);              %   replaces the element with the greater of
                                       the 2 arguments
                                   %   if element > av, replace with diff, else
                                       replace with 0
   for ctr=smoothmin:smoothmax     %   defines range of row elements on which
                                       window centered
      ss=0;
      for dsm = -smooth:smooth     %   add all of the values inside the window
                                       together
         ss=ss+row(ctr+dsm);       %   note: all original values that were < av
                                       are now 0
      end
      row(ctr)=row(ctr)-ss/(smooth+1);     %  subtract the "new avg" from the
                                                original value of window center
   end

   row=max(row,0);                 %   sets all negative values equal to 0
   av=mean(row);                   %   computes average of "new" row
   st=std(row);                    %   standard deviation
   row=max(row-av-10*st,0);        %   result:  row element that is > 10 std dev
                                       above average
                                   %   will be positive, else it will be set to 0
   corim(481-i,:)=row;             %   builds matrix from each "row" in rev order
                                       from bottom to top of image array
end
[I1,J1] = find(corim > 0);
I1 = 480*ones(length(I1),1)-I1;    %  converts row coordinates in uv to ij
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fin=zeros(points,4);

for i=1:points
   [colmax,rowloc]=max(corim);       %   colmax: max value for each col;
                                     %   rowloc: row index of max value of each
                                         column
   [fin(i,4),fin(i,2)]=max(colmax); %   place max value of colmax in ith row,
                                         4th col of fin
                                     %   place col index of colmax max in ith
                                     %   row, 2nd col of fin
   fin(i,1)=rowloc(fin(i,2));        %   place row index of colmax max in ith
                                         row, 1st col of fin
   mma=max(fin(i,1)-spot,1);         %   these 4 lines define a box around a
                                     %   bright point,
   mmi=min(fin(i,1)+spot,ypic);      %   inside which pixel intensities are set
                                     %    to zero
   mla=max(fin(i,2)-spot,1);
   mli=min(fin(i,2)+spot,xpic);

   for yl=mma:mmi                % this loop sets equal to zero the pixels in
                                 % "corim" that are  inside the box defined above
      for xl=mla:mli
         corim(yl,xl)=0;
      end
   end
end

centeru = fin(:,2);     %  extracts column coordinates of 5 points in uv format
```

```
centerv = fin(:,1);     %  extracts row coordinates of 5 points in uv format
I2 = 480*ones(points,1)-centeru;  %  converts row coordinates in uv to ij
J2 = centerv;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for ii=1:points     %  adds the distances from pt ii to all other points
                    %  together and stores them.  The points with the 3
   for jj=1:points  %  smallest sums are points of interest.
      fin(ii,3)= fin(ii,3)+sqrt((fin(ii,1)-fin(jj,1))^2+(fin(ii,2)-
                 fin(jj,2))^2);
   end
end
fin = sortrows(fin,3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
centeru = fin(1:3,2);  %  extracts column coordinates of 3 points in uv format
centerv = fin(1:3,1);  %  extracts row coordinates of 3 points in uv format
centeri = 480*ones(3,1)-centerv;    %  converts row coordinates in uv to ij
centerj = centeru;
center = [centeri  centerj];        %  ij coordinates of 3 spots of interest
center = sortrows(center);
fprintf(['\n Time to locate 3 hotspots:      ', num2str(toc-start),' seconds
\n'])

igrowth = .3;
jgrowth = .3;
newbox = nextbox(center,igrowth,jgrowth);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3. DEFINEBOX.M

```
function  newbox = definebox(I, filename, img_no, filetype)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Read image from file

%I = imread([filename,num2str(img_no),filetype]);     %read in image
%I = rgb2gray(I);       % makes image array compatible dimension for
                                      % image processing functions
figure
   imcontour(I,10)
   grid on
   title([filename,num2str(img_no),filetype])
   xlabel('j'); ylabel('i')
   zoom on
   fprintf('\nPlace a zoom box around the points of interest.\n')
   fprintf('Hit return when done.\n\n')
   pause
   j = round(get(gca,'XLim'));
   i = round(get(gca,'YLim'));
   imin = i(1);
   imax = i(2);
   jmin = j(1);
   jmax = j(2);
   close
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
newbox = [imin, imax, jmin, jmax];
show_peaks(I, filename, filetype, img_no, imin, imax, jmin, jmax);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4. FILTERBOX.M

```
function newbox = filterbox(I, filename, img_no, filetype, vtol, htol)

clc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  FILTER IMAGE
%
fprintf('\n Filtering first image. \n')
start = toc;
I = imfilter(I);
fprintf(['\n Time to filter first image:      ', num2str(toc-start),' seconds
\n'])

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

threshold = .45;
imin = 1;
imax = 480;
jmin = 1;
jmax = 640;
box = [imin imax jmin jmax];
igrowth = .3;
jgrowth = .3;


fprintf('\n Determining location of spots in first image and establishing first
search box. \n')
boxstart = toc;
[spots,polycenter,diff] = findxy(I, box, threshold, vtol, htol);
%plotstem(filename, img_no, diff, imin, imax, jmin, jmax, filetype)
newbox = nextbox(polycenter,igrowth,jgrowth);
boxstop = toc;
fprintf(['\n Time required to compute first search box: ',num2str(boxstop-
boxstart),' seconds. \n'])
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5. FINDCENTER.M

```
function [brightpoints,center,threshold,vtol,htol,centroidtime] =
findcenter(threshold,I,box,vtol,htol,centroidtime)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  INITIALIZATION
%
imin = box(1);
imax = box(2);
jmin = box(3);
jmax = box(4);
vtol0 = 6;
htol0 = 4;
spots = 0;    %  counter, a spot consists of all pixels w/in box defined by vtol
              %  & htol
lothresh = .4;          % initial minimum threshold
hithresh = 1;           % initial maximum threshold
```

```
%threshold = .9;        % initial threshold
laps = 0;               % prevents auto decrease of threshold first time in loop
C = zeros(180,2,20);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while (spots ~= 3)
   C(:,:,1:3)=0;
   %  ADJUST THRESHOLD IN RESPONSE TO NUMBER OF SPOT %%%%%%%%%%%%%%%%%%%%%%%%%%
   if (spots < 3 & laps > 0)      %  decrease threshold, will not enter first
                                  %  time in loop
      hithresh = threshold;       % when 'spots' is always less than 3
      threshold = (threshold + lothresh)/2;
   end % if
   if spots > 3    % increase threshold
      lothresh = threshold;
      threshold = (hithresh + threshold)/2;
   end % if
   if (threshold - lothresh <= 0.005) | (hithresh - threshold <= 0.005)
                                                    % prevent inf loop
      fprintf('\n * * * * * Unable to isolate 3 points. Increasing spot
                  tolerance. * * * * *\n\n')
      vtol = vtol + vtol0
      htol = htol + htol0
      if vtol >100 | htol > 50
         vtol = vtol0;           % reset vtol to initial value
         htol = htol0;           % reset htol to initial value
         break
      end % if
   end % if

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   laps = 1;
   starttime = toc;     % time hack to note start of loop at a given threshold
   brightpoints = [];   % all points in Izoom that exceed threshold
   center = [];         % array of spot centroids

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %  FIND ALL PIXELS IN Izoom WHOSE INTENSITY EXCEEDS THRESHOLD
   Izoom = I(imin:imax, jmin:jmax);          %  portion of original image, I,
                                             %  inside zoom box
   [row,col] = find(Izoom > threshold);      %  find all pixels in Izoom that
                                             %  exceed threshold
   while isempty(row)       % decrease threshold if no points found in Izoom
      hithresh = threshold;
      threshold = (threshold + lothresh)/2;
      [row,col] = find(Izoom > threshold);
   end % while      %  exit when at least 1 pixel found in Izoom that exceeds
                    %  current threshold

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   hotpoints = sortrows([row,col]);          %  row sort of pixels that exceed
                                             %  current threshold
   morepoints       = 1;                     %  BOOLEAN:  1 = true, 0 = false
   spots            = 0;                     %  reset spot counter to zero
   brightpoints(:,1) = hotpoints(:,1) + imin - 1;
                                 % convert local(Izoom) index to image index
   brightpoints(:,2) = hotpoints(:,2) + jmin - 1;
                                 % convert local (Izoom) index to image index
   spotdim = [];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %  SEPARATE PIXELS IN hotpoints INTO SPOTS
   while (morepoints == 1) & (spots <= 3)  %  TRUE if there are hotpoints that
                                           %  don't yet belong to a spot
      spots = spots + 1;
      morepoints = 0;          %  reset FALSE until pixels are found that don't
                               %  belong to a spot
      hotptsize = size(hotpoints);
      no_rows = hotptsize(1,1);
      firstpoint = hotpoints(1,:);
      currentspot = firstpoint;
      holdingspot = [];        %  used to "collect pixels" outside vtol/htol of
                               %  current spot
      thisrow = 1;
      nextrow = thisrow + 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %   DETERMINE IF PIXEL BELONGS IN currentspot OR PUT IN holdingspot TO ASSIGN
       TO OTHER SPOT
      while (nextrow <= no_rows) & (spots <= 3)
         thisi  = hotpoints(thisrow,1);
         thisj  = hotpoints(thisrow,2);
         thispoint = hotpoints(thisrow,:);
         nexti = hotpoints(nextrow,1);
         nextj = hotpoints(nextrow,2);
         nextpoint = hotpoints(nextrow,:);
         idiff = abs(nexti - thisi);
         jdiff = abs(nextj - thisj);
         if (idiff <= vto1 & jdiff <= hto1)
            currentspot = [currentspot; nextpoint];
            thisrow = nextrow;
            nextrow = thisrow + 1;
         else
            morepoints = 1;
            holdingspot = [holdingspot; nextpoint];
            nextrow = nextrow + 1;
         end % if/else
      end % while       % exit when all remaining pixels have been evaluated

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      [r,c] = size(currentspot);
      C(1:r,1:c,spots) = currentspot;   %  buffer to hold each spot in
                                        %  individual 2x2 array
      hotpoints = holdingspot;   %  reassign "leftover" pixels to hotpoints for
                                 %  assignment to new spot
   end %while    %  exit when there are no more pixels that need to be assigned
                 %  to a spot

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end % while

for s = 1:spots
   [r,c] = find(C(:,:,s) ~= 0);
   center(s,:) = weightedcenter(Izoom,C(1:max(r),1:2,s)) + [imin-1 jmin-1];
%compute centroid
end
center = sortrows(center);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

6. FINDDIFF.M

```
function [spots, diff] = finddiff(I,threshold,box)

imin = box(1);
imax = box(2);
jmin = box(3);
jmax = box(4);

fit_order = 2;

[row,col] = size(I);
jstart = jmin + 10;
jend = jmax - 10;
spots = [];
diff = [];
for i = imin:imax
   ifar = I(i,:);
   j = jstart:jend;
   p = polyfit(j,ifar(j),fit_order);
   f = polyval(p,j);
   % insert Plot Routine 1 here if desired

   [y,x] = max(ifar(j)-f);
   jpeak = x + jstart;
   diff = [diff; i jpeak y];
   if y > threshold
      spots = [spots; i jpeak];
   end %if
end %for
%hold off

istart = imin;
iend = imax;

for j = jmin:jmax
   jfar = I(:,j);
   i = istart:iend;
   p = polyfit(i,jfar(i)',fit_order);
   f = polyval(p,i);
   % insert Plot Routine 2 here if desired

   [y,x] = max(jfar(i)'-f);
   ipeak = x + istart;
   diff = [diff; ipeak j y];
   if y > threshold
      spots = [spots; ipeak j];
   end %if
end %for
spots = sortrows(spots);
```

```
% Plot Routine 1

%   if i == 42
%      figure(i)
%         plot(j,ifar(j),'k+',j,f,'k.-')
%         axis([jstart jend 0 1])
%         xlabel('Pixel Column')
%         ylabel('Relative Intensity')
%         title(['Row: ',num2str(i),', Fit order: ',num2str(fit_order)])
%         legend('actual data','curve fit')
%         grid on
%         hold on
%   end %if, Plot Routine 1


% Plot Routine 2

%  if j == 204
%     figure(j)
%        plot(i,jfar(i),'bd',i,f,'r*-')
%        grid on
%        axis([istart iend 0 1])
%        title(['Column: ',num2str(j),', Fit order: ',num2str(fit_order)])
%  end %if
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

7. FINDXY.M

```
function [spots,center,diff] = findxy(I, box, threshold, vtol, htol)

[hotpoints,diff] = finddiff(I,threshold, box);

imin = box(1);
imax = box(2);
jmin = box(3);
jmax = box(4);

morepoints       = 1;   %BOOLEAN:  1 = true, 0 = false
spot             = 0;
spots = hotpoints;

while morepoints == 1   %true
   spot = spot + 1;
   morepoints = 0;
   hotptsize = size(hotpoints);
    no_rows = hotptsize(1,1);
    firstpoint = hotpoints(1,:);
    currentspot = firstpoint;
    holdingspot   = [];
   thisrow = 1;
   nextrow = thisrow + 1;
   while (nextrow <= no_rows)
    thisi  = hotpoints(thisrow,1);
    thisj  = hotpoints(thisrow,2);
    thispoint = hotpoints(thisrow,:);
    nexti = hotpoints(nextrow,1);
    nextj = hotpoints(nextrow,2);
    nextpoint = hotpoints(nextrow,:);
    idiff = abs(nexti - thisi);
```

```
      jdiff = abs(nextj - thisj);
      if (idiff <= vtol & jdiff <= htol)
         currentspot = [currentspot; nextpoint];
         thisrow = nextrow;
         nextrow = thisrow + 1;
      else
         morepoints = 1;
         holdingspot = [holdingspot; nextpoint];
         nextrow = nextrow + 1;
      end % if/else
   end % while
   [r,c] = size(currentspot);
   C(1:r,1:c,spot) = currentspot;%  buffer to hold each spot in individual 2x2
                                 %  array
   hotpoints = holdingspot;      %  zeroize currentspot
end %while
for s = 1:spot
   [r,c] = find(C(:,:,s) ~= 0);
   center(s,:) = weightedcenter(I,C(1:max(r),1:2,s)) + [imin-1 jmin-1];
%compute centroid
end
center = sortrows(center);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

8. IJ2UV.M

```
% This function transforms a 2-column matrix of ij coordinates to a
% 2-column matrix of uv coordinates.  It assumes imax = 480 and jmax = 640.

function uv = ij2uv(ij)

[r,c] = size(ij);

uv(:,1) = ij(:,2) - 320*ones(r,1);
uv(:,2) = 240*ones(r,1) - ij(:,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

9. IMFILTER.M

```
function If = imfilter(In)

[r,c] = size(In);
% Weibull Distribution
x = 1:r;
lambda = 1.8;       % larger values decrease variance
sigma = 60;
fW = (((lambda*x)/(sigma^2)).^(lambda-1)).*exp(-((x/sigma).^lambda));
fW = fW/max(fW);
W = diag(fW);

% Gaussian Distribution - apply to columns
x = 1:c;
sigma = 30;
mu = 220;
fN = (1/(sigma*sqrt(2*pi)))*exp(-0.5*((x-mu)/sigma).^2);
fN = fN/max(fN);
G = diag(fN);
```

```
% Composite Filter
If = W'*In*G;

If = mat2gray(If);

%  folder            Picture  lambda(W)   sigma(W) sigma(G)    mu(G)
%
%  time1747...       1        1.8         100      75          320
%  turning           4        2           40       30          170
%  flt1_174505...    25       1.8         60       30          220
%%%
```

12. PLOTBOXHEIGHTWIDTH.M

```
function plotboxheightwidth(box,no_imgs,first_img)

index = first_img:(first_img + no_imgs - 1);
di = box(:,2)-box(:,1);
dj = box(:,4)-box(:,3);

figure
   plot(index,dj,'k+',index,di,'k.')
   grid on
   axis([first_img (first_img + no_imgs - 1) 0 max(dj)])
   xlabel('Frame Number')
   ylabel('Pixels')
   title('Search Box Dimension Size')
   legend('width','height',2)
%%%
```

```
I = [i1 i2 i3];
J = [j1 j2 j3];
Imin = [];
Imax = [];
Jmin = [];
Jmax = [];
for p = 1:length(i1)
   Imin = [Imin; min(I(p,:))];
   Imax = [Imax; max(I(p,:))];
   Jmin = [Jmin; min(J(p,:))];
   Jmax = [Jmax; max(J(p,:))];
end % for

delI = Imax - Imin;
delJ = Jmax - Jmin;

delImin = -diff(Imin)./delI(1:(length(delI)-1));
delImax = diff(Imax)./delI(1:(length(delI)-1));
delJmin = -diff(Jmin)./delJ(1:(length(delJ)-1));
delJmax = diff(Jmax)./delJ(1:(length(delJ)-1));
x = frame(1):length(delImin)+frame(1)-1;


figure
subplot(2,1,1)
   plot(x,100*delImin,'b',x,100*delImax,'r')
   grid on
   axis([frame(1) length(delImin)+frame(1)-1 -igrowth*100 igrowth*100])
   title('Expansion of box top/bottom boundaries as % of (i_m_a_x - i_m_i_n)')
   legend('top','bottom',-1)
   ylabel('Percent Expansion')
   xlabel('Frame')
subplot(2,1,2)
   plot(x,100*delJmin,'b',x,100*delJmax,'r')
   grid on
   axis([frame(1) length(delImin)+frame(1)-1 -jgrowth*100 jgrowth*100])
   title('Expansion of box left/right boundaries as % of (j_m_a_x - j_m_i_n)')
   legend('left','right',-1)
   ylabel('Percent Expansion')
   xlabel('Frame')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

14. PLOTCENTROID.M

```
function p = plotcentroid( filename, filetype, img_no, spots, box, threshold,
                           center)
imin = box(1);
imax = box(2);
jmin = box(3);
jmax = box(4);

figure(img_no)
      plot(spots(:,2),spots(:,1),'k.')
      axis ij
      axis([jmin jmax imin imax])
      grid on
      title(['Hotspot Centroid Location,  File: ',
filename,num2str(img_no),filetype,'
             ','Threshold: ',num2str(threshold)])
      xlabel('Pixel Column (j)')
      ylabel('Pixel Row (i)')
      hold on
         plot(center(:,2),center(:,1), 'k+')
      hold off
      legend('> threshold', 'centroid',1)
%%%
```

17. PLOTTHRESHOLD.M

```
function plotthreshold(center, first_img, no_imgs)

figure
   plot(center(:,1),center(:,8),'k.')
   grid on
   xlabel('Frame number')
   ylabel('Threshold')
   axis([first_img (first_img + no_imgs-1) 0 1])
   title('Final 3-point Threshold')
%%%
```

20. SHOWBRIGHTEST.M

```
function showbrightest(filename, img_no, filetype, centerj, centeri)

I = imread([filename, num2str(img_no), filetype]);   % read image from file

figure
   imshow(I)
   hold
   plot(centerj,centeri,'r+')
   axis ij
   axis([0 640 0 480])
   grid on
   title(['Five Brightest Spots in ',filename, num2str(img_no), filetype])
   legend('bright spot')
%%%
```

22. SORTIJ123.M

```
function centerij123 = sortij123(centerij)

[r,c] = size(centerij);

for row = 1:r
   abc = [  centerij(row,2:3);
            centerij(row,4:5);
            centerij(row,6:7)];
   i = find(abc(:,2) == min(abc(:,2))); %  find element of abc that has minimum
                                        %  value of j
   pt3 = abc(i,1:2);
   pt12 = abc(find(abc(:,2) ~= pt3(2)),1:2);
   i = find(pt12(:,1) == max(pt12(:,1)));
   pt1 = pt12(i,1:2);
   pt2 = pt12(find(pt12(:,1) ~= pt1(1)),1:2);
   centerij(row,2:7) = [pt1 pt2 pt3];
end %for

centerij123 = centerij;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

23. WEIGHTEDCENTER.M

```
function p = weightedcenter(I,spot)

[row,col] = size(spot);
for r = 1:row
   brightness(r) = I(spot(r,1),spot(r,2));
   wspot(r,:) = spot(r,:)*brightness(r);
end %for
p(1,1) = sum(wspot(:,1))/sum(brightness);
p(1,2) = sum(wspot(:,2))/sum(brightness);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Doctor Isaac I. Kaminer, Code AA/KA 3
   Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA 93943-5121

4. Doctor Oleg A. Yakimenko, Code AA/YA 3
   Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA 93943-5121

5. Doctor Russell Duren, Code AA/DR 1
   Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA 93943-5121

6. Doctor Conrad F. Newberry, Code AA/NE 1
   Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA 93943-5121

7. Department of Aeronautics and Astronautics 1
   Code AA
   Naval Postgraduate School
   Monterey, CA 93943-5106

8. Lieutenant Commander Paul A. Ghyzel, USN 2
   14815 Cranoke Street
   Centreville, VA 20120

63 290NPG TH 2588
6/02 22527-200 NLB

DUDLEY KNOX LIBRARY
3 2768 00402333 3